3. Check Code: Customizing the Data Entry Process

Introduction

The Check Code tool allows users to customize the data entry process. It is useful to check for errors during the data entry process, to do automatic calculations in fields, and to skip over parts of the questionnaire if certain conditions are met. Check Code makes it possible to instruct the Enter tool to perform such operations automatically. By using Check Code, you can protect data against many common types of errors by setting rules for data entry. Check Code is created using the Check Code Editor.

Examples of operations that can be performed in Check Code include:

- Displaying messages that appear to be part of the questionnaire
- Calculating fields from mathematical operations
- Checking one or more fields for relationships (e.g., making sure birth date is earlier than current date)
- Checking fields for inconsistencies (e.g., male pregnancies)
- Displaying error messages due to improper entries in a field
- Complex statistics or other operations that are written in other languages
- Automatic indexing of fields for faster searching
- Automatic searches during data entry

An operation can be performed automatically each time data is entered in a field or conditionally when a certain value is entered. Check code is optional and is designed to allow the user to provide customized data entry processes.

Accessing Check Code Program Editor

To navigate to the **Check Code Program Editor**, click the **Check Code button** in the Form Designer tool bar after opening your Epi Info 7 project.

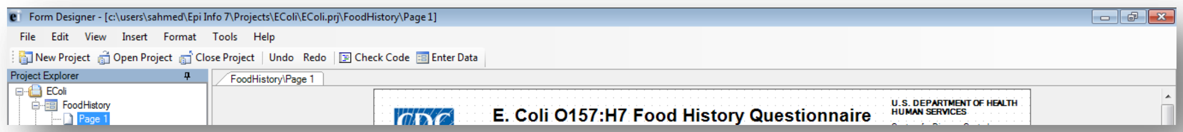

Form Designer - [c:\users\sahmed\Epi Info 7\Projects\EColi\EColi.prj\FoodHistory\Page 1]

Figure 3.1: Check Code button

Alternatively, select **Tools > Check Code Editor** from the **Form Designer** navigation menu.

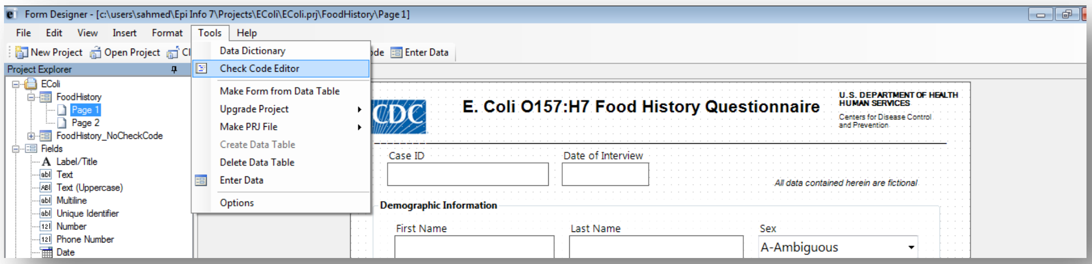
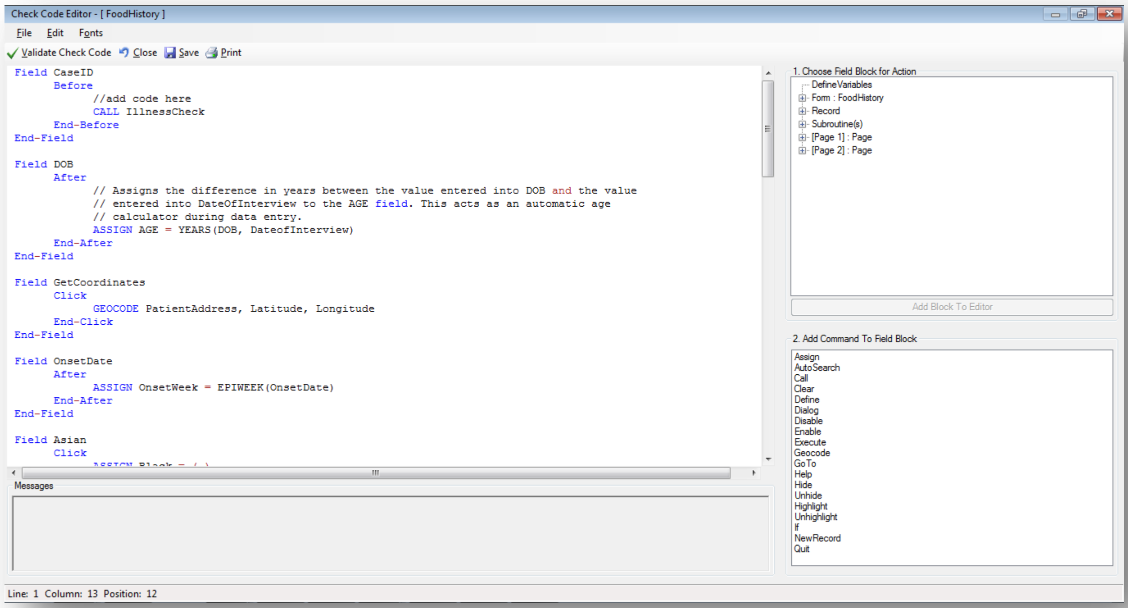


Figure 3.2: Form Designer Menu

Navigate the Check Code Program Editor

The Check Code Editor window contains four sections:

- **Choose Field Block for Action**
- **Add Command to Field Block**
- **Program Editor**
- **Messages**



```
Field CaseID
      Before
            //add code here
            CALL IllnessCheck
      End-Before
End-Field

Field DOB
      After
            // Assigns the difference in years between the value entered into DOB and the value
            // entered into DateOfInterview to the AGE field. This acts as an automatic age
            // calculator during data entry.
            ASSIGN AGE = YEARS(DOB, DateofInterview)
      End-After
End-Field

Field GetCoordinates
      Click
            GEOCODE PatientAddress, Latitude, Longitude
      End-Click
End-Field

Field OnsetDate
      After
            ASSIGN OnsetWeek = EPIWEEK(OnsetDate)
      End-After
End-Field

Field Asian
      Click
            ASSIGN Black = (.)
```

Figure 3.3: Check Code Program Editor

1. The **Choose Field for Block Action** tree allows you to select fields and sets when the actions designated by the Check Commands occur during data entry.

2. The **Add Command to Field Block** window displays all the available check commands used in the Form Designer program.

3. The **Program Editor** window displays the code generated by the commands created from the **Choose Field Block for Action** or **Add Command to Field Block** window. Code can also be typed and saved directly into the Program Editor.

4. The **Message** window alerts you of any check command problems

There are several options on the toolbar at the top left that allow you to save, edit, validate and change the font of your program editor. You can close the Check Code Editor and return to your form by clicking on the red X button at the top right of your screen, clicking on the Close button, or by pressing the F10 key.

Check Code Commands

Check commands are structured in blocks. Each block begins with a field, page or form name and ends with the word END. All commands must be within a field-name block. Commands in a block are usually activated either before or after an entry is made in the field. For some field types, blocks can also be activated when clicking on the control (i.e. checkboxes and command buttons). The usual case is that the commands are performed after an entry has been made and the user has pressed the <ENTER> key or when the cursor has left the field automatically. This behavior can be altered by placing commands in blocks called BEFORE. The format of check code blocks is usually structured as follows:

```
Field VARIABLENAME
After
Check Code syntax inserted here
End-After
End-Field
```

- The **FIELD** parameter establishes to which field name the check code block corresponds.
- The **AFTER** parameter specifies when the action will occur. The AFTER event is executed as soon as the cursor leaves that field.
- The **END-AFTER** parameter specifies the closing of the commands to be executed, in this example, for the AFTER event. In other words, any check code inserted between the AFTER and END-AFTER section will be executed for that field after the cursor leaves the field.

- The **END-FIELD** parameter simply closes the block of commands incorporated for the corresponding field.

In the example below, we have incorporated an AFTER event for a field called DOB. The block of check code will execute the assignment of a value to the field AGE using the YEARS function. The YEARS function will calculate the difference between two date fields and provide the result in Years.

```
Field DOB
After
ASSIGN AGE=YEARS(DOB,SYSTEMDATE)
End-After
End-Field
```

Basic Check Code Command Rules

1. Check Commands must be placed in a block of commands corresponding to a variable/field in the database. Special sections are provided to execute commands before or after you display a form, page, or record.
2. Comments preceded by two forward slashes ("//") may be placed within blocks of commands and will be ignored during execution of check code.
3. Commands in a block are activated before or after you make an entry in the field. By default, commands are performed after an entry has been completed with <Enter>, <PgUp>, <PgDn>, or <Tab>, or another command causes the cursor to leave the field (e.g., GOTO). Commands can also be activated when clicking on a field or when selecting a value from a drop down list (versions 7.1.4 or higher only).
4. Check commands for each field are stored in the form in a record associated with a particular field.
5. Commands are inserted automatically through interaction with the dialog boxes. The syntax generated by the dialogs is then displayed in the Check Code Editor. Text can also be edited and saved in the Program Editor.
6. BEFORE and AFTER commands can be inserted into fields but also into a form, page or a record.

Creating a Check Code Block

1. From **Choose Field Block for Action**, select and expand the form, page, record, or field that will receive the commands. Expand by clicking the + sign.

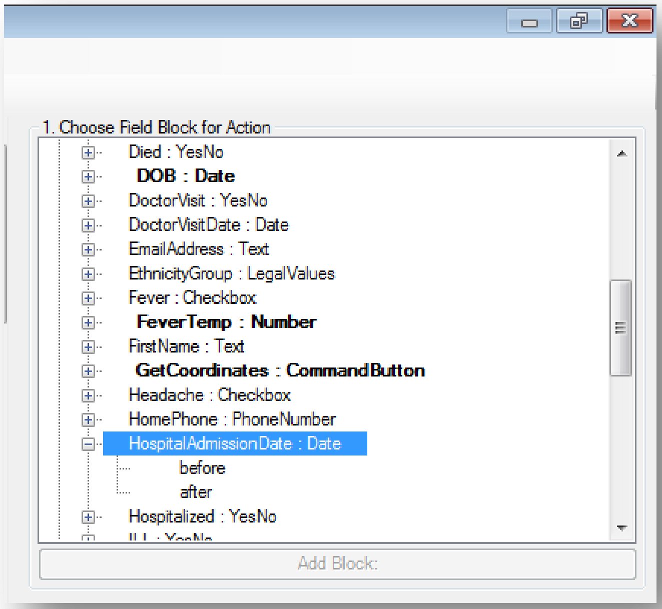

1. Choose Field Block for Action
Died : YesNo
DOB : Date
DoctorVisit : YesNo
DoctorVisitDate : Date
EmailAddress : Text
EthnicityGroup : LegalValues
Fever : Checkbox
FeverTemp : Number
FirstName : Text
GetCoordinates : CommandButton
Headache : Checkbox
HomePhone : PhoneNumber
HospitalAdmissionDate : Date
before
after
Hospitalized : YesNo

Figure 3.4: Choose Field Block for Action

2. Select whether the command will be executed **before** or **after** data entry into the form, page, record, or field by clicking in the corresponding option (in this example AFTER).

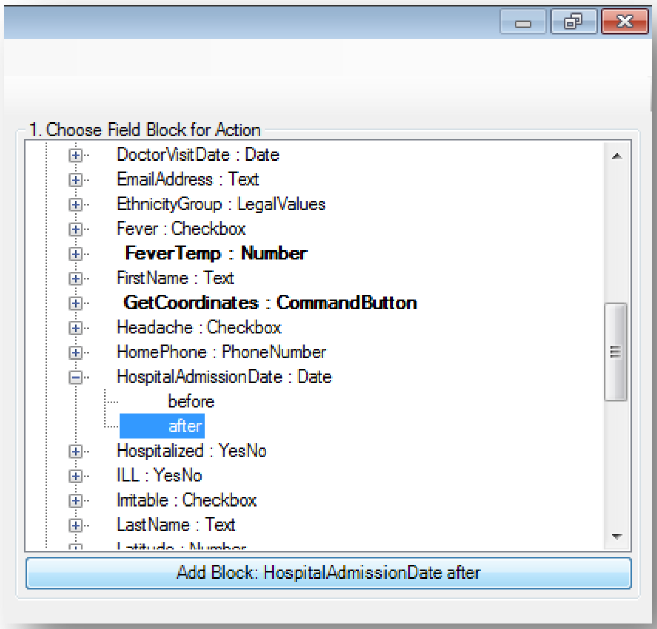
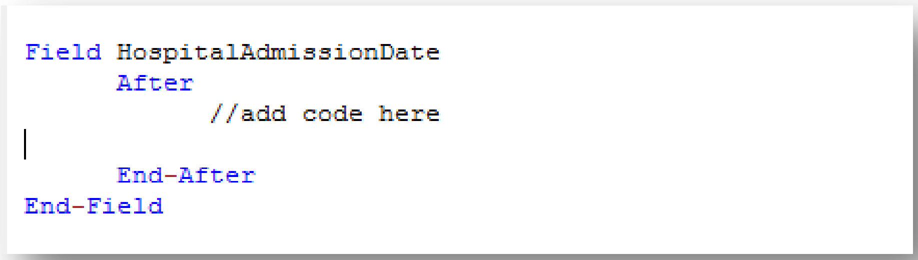

1. Choose Field Block for Action
DoctorVisitDate : Date
EmailAddress : Text
EthnicityGroup : LegalValues
Fever : Checkbox
FeverTemp : Number
FirstName : Text
GetCoordinates : CommandButton
Headache : Checkbox
HomePhone : PhoneNumber
HospitalAdmissionDate : Date
before
after
Hospitalized : YesNo
ILL : YesNo
Irritable : Checkbox
LastName : Text

Figure 3.5: Add Block

3. You can either double click on the AFTER event or click the **Add Block** button in order to insert the block. The Check Code Block appears in the **Check Code Editor**.



```
Field HospitalAdmissionDate
      After
            //add code here

      End-After
End-Field
```

Figure 3.6: Code Block Added to Editor

After a Check Code Block has been created for a form, page, record, or field, you can insert commands within the block using the **Add Command to field Block** section.

Create a Skip Pattern with GOTO

You can create skip patterns by changing the tab order and setting a new cursor sequence in a form, or by creating Check Code using the **GOTO** command. Skip patterns can also be created based on the answers to questions using an **IF/THEN** statement. In the following example, we will add check code to move the cursor to the **Ethnicity** field after data is entered into the **DOB** field.

1. From Form Designer, Open the **Ecoli.prj project**.
2. Double click on the **FoodHistory_NoCheckCode** form.
3. Click **Check Code** or select **Tools > Check Code Editor**. The Check Code Editor opens.
4. From the **Choose Field Block for Action section,** expand the node for Page 1 to see the various fields on the first page.
5. Expand the node for the field **DOB**.

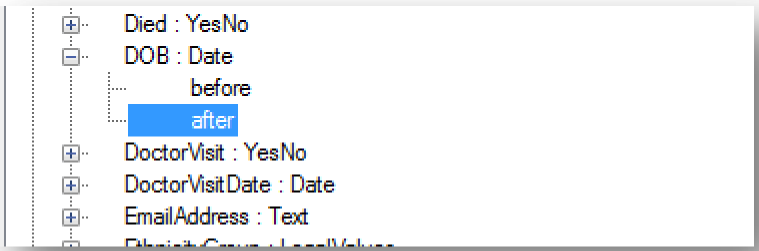
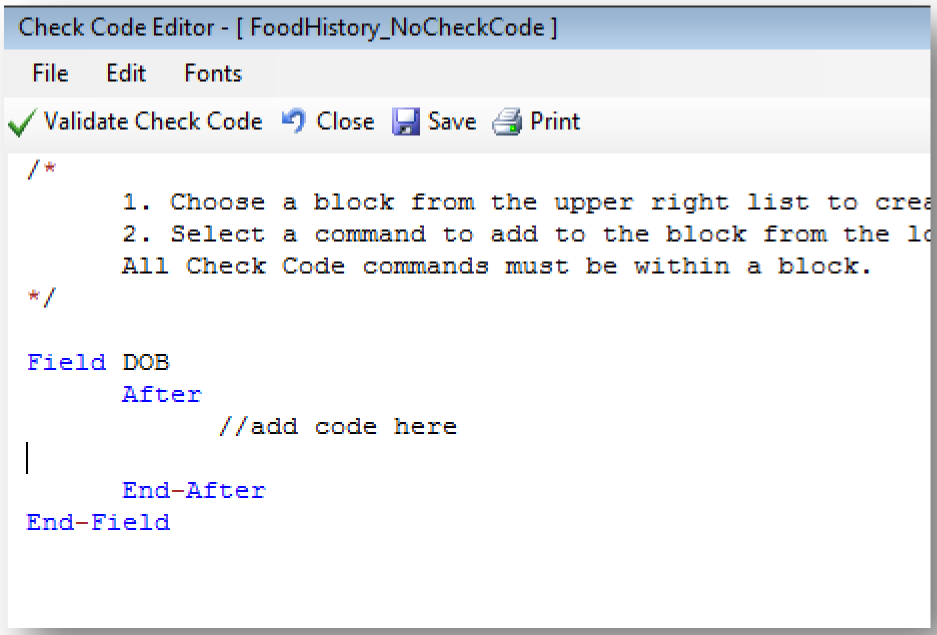


Figure 3.7: Dialog Block for Action after DOB

6. Double click on the **after** event.
7. A block of code for the **DOB** field will display in the **Check Code Editor**.



```
/*
        1. Choose a block from the upper right list to crea
        2. Select a command to add to the block from the lo
        All Check Code commands must be within a block.
*/

Field DOB
        After
                //add code here

        End-After
End-Field
```

Figure 3.8: Check Code Block for Action After DOB

8. Click **GoTo** from the **Add Command to Field Block** list box. The **GOTO** dialog box opens.
9. Select the **EthnicityGroup** field for the cursor to jump into after data has been entered in the **DOB** field. The code will run after the cursor leaves the field.

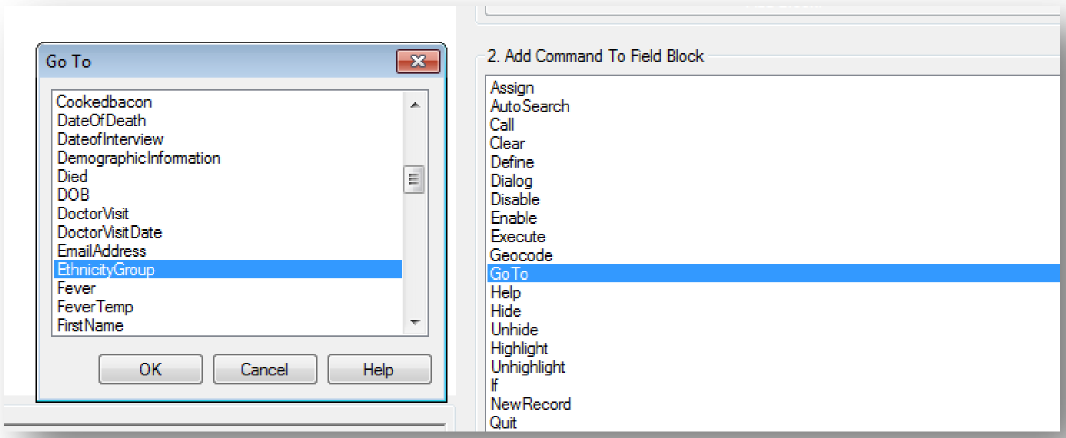
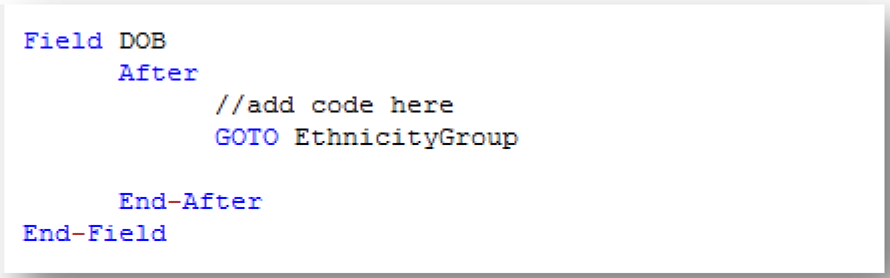


Figure 3.9: Skip Pattern **Go To** dialog box

10. Click **OK**. The code appears in the Check Code Editor.



```
Field DOB
      After
            //add code here
            GOTO EthnicityGroup

      End-After
End-Field
```

Figure 3.10: Skip Pattern Check Code Command

11. Click the **Verify Check Code** button from the Check Code Editor.
12. Click the **Save** button from the Check Code Editor.
13. Click **Close** to return to the form.

To test the skip pattern, open the form in the Enter Data tool. Use the **tab key** to ensure that upon leaving the field with the **GOTO** command, the cursor goes to the specified field.

Create a Skip Pattern Using IF/THEN and GOTO

Use **IF/THEN** statements to create skip patterns based on the answers to questions in the form. This example creates code, which states that if the person answered No (-) for the **Hospitalized** field, then the cursor subsequently jumps to the field **Was the patient treated with antibiotics?** and skips the **Hospital Admission** date field.

1. From **Form Designer**, Open the **Ecoli.prj project**.
2. Double click on the **FoodHistory_NoCheckCode** form.
3. Click **Check Code** or select **Tools > Check Code Editor**. The Check Code Editor opens.
4. From the **Choose Field Block for Action section,** expand the node for Page 1 to see the various fields on the first page.
5. Expand the node for the **Hospitalized** field.
6. The action needs to occur after data are entered into the **Hospitalized** field.
7. Double click on the **after** event.
8. A block of code for the **Hospitalized** field is created and displayed in the Check Code Editor.

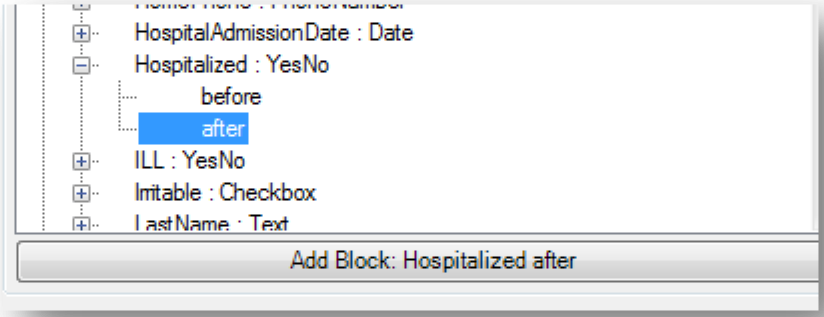
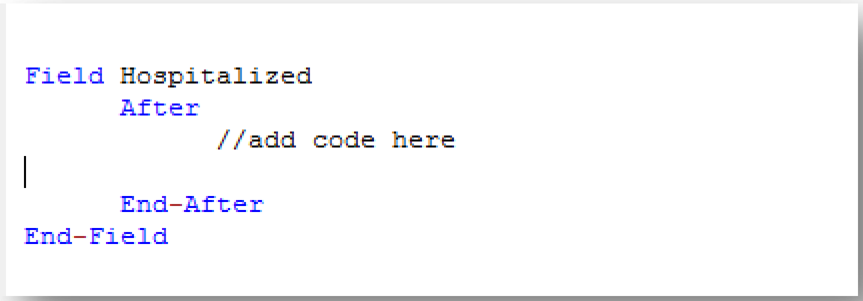


Figure 3.11: Dialog Block for Action after Hospitalized

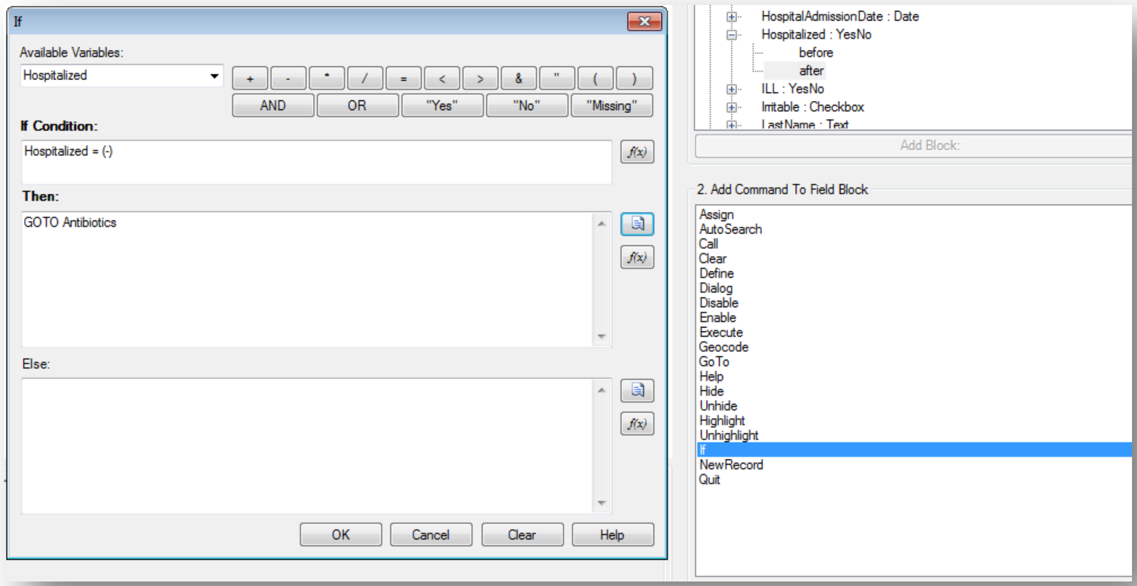
9. The code appears in the Check Code Editor.



```
Field Hospitalized
      After
            //add code here

      End-After
End-Field
```

Figure 3.12: If/Then Block Code After Hospitalized

10. Click **If** from the **Add Command to Field Block** list box. The **IF** dialog box opens.
11. From the **Available Variables** drop-down list, select the **field** to contain the action. For this example, select **Hospitalized**. The selected variable appears in the **If Condition** field.
12. From the Operators, click **=**.
13. From the Operators, click **No**. The **If Condition** field will read Hospitalized=(-).
14. Click the **Code Snippet** button in the *Then* section. A list of available commands appears.
15. From the command list, select **GoTo**. The **GOTO** dialog box opens.
16. Select the **field** for the cursor to jump to based on a *No* answer from the list of variables. For this example, select **Antibiotics**.
17. In the **GOTO** dialog box, click **OK** to return to the **IF** dialog box.



If Condition:

Hospitalized = (-)

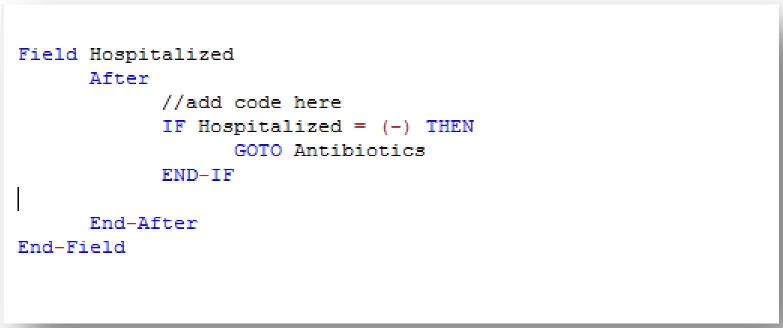
Then:

GOTO Antibiotics

Else:

Figure 3.13: If/Then dialog box

18. Click **OK**. The code appears in the Check Code Editor. The example code appears as:



```
Field Hospitalized
      After
            //add code here
            IF Hospitalized = (-) THEN
                  GOTO Antibiotics
            END-IF

      End-After
End-Field
```

Figure 3.14: If/Then Check Code Command

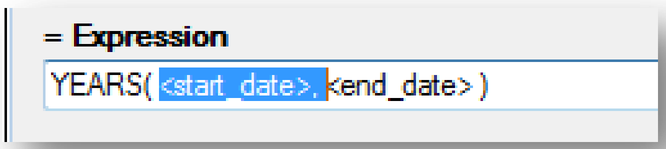
19. Click the **Verify Check Code** button from the Check Code Editor.
20. Click the **Save** button from the Check Code Editor.

Using Functions with a Date field

To program a mathematical function, use the Program Editor and the **ASSIGN** command. For example, Check Code can be created to automatically calculate the age of a respondent based on the date of birth and the date the form was completed, or the system date of the computer when data were entered.

This example uses a field called **DateOfBirth** and a field called **Age** from the **FoodHistory_NoCheckCode** form in the **Ecoli** project to demonstrate the use of the **ASSIGN** command and the function **YEARS**.

1. Click **Check Code** or select **Tools > Check Code Editor**. The Check Code Editor opens.
2. From the **Choose Field Block for Action** section, expand the node for the page where the **DateofBirth** field is located.
3. Expand the node for the **DateofBirth** field.
4. Double click on the **after** event.
5. A block of code for the **DateofBirth** field is created and displayed in the Check Code Editor.
6. Click **Assign** from the **Add Command to Field Block** list box. The **Assign** dialog box opens.
7. From the **Assign Variable** drop-down list, select the field where the calculated value should appear. For this example, select the **Age** field.
8. Click on the functions button .
9. Select the **Date Functions** option.
10. Click on the **YEARS** function.
11. Double click on the <**start_date**> parameter. This will highlight that section of the command.

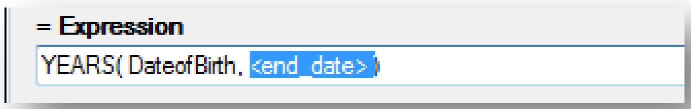


= Expression

YEARS(<start_date>, <end_date>)

Figure 3.15: Start Date Parameter

12. Select **DateOfBirth** from the list of Available Variables
13. Double click on the <**end_date**> parameter. This will highlight that section of the command.



= Expression

YEARS(DateofBirth, <end_date>)

Figure 3.16: End Date Parameter

14. Click on the functions button .
15. Select the **System Functions** option.
16. Click the **SYSTEMDATE** function. Once completed, the syntax will be inserted on the dialog window.

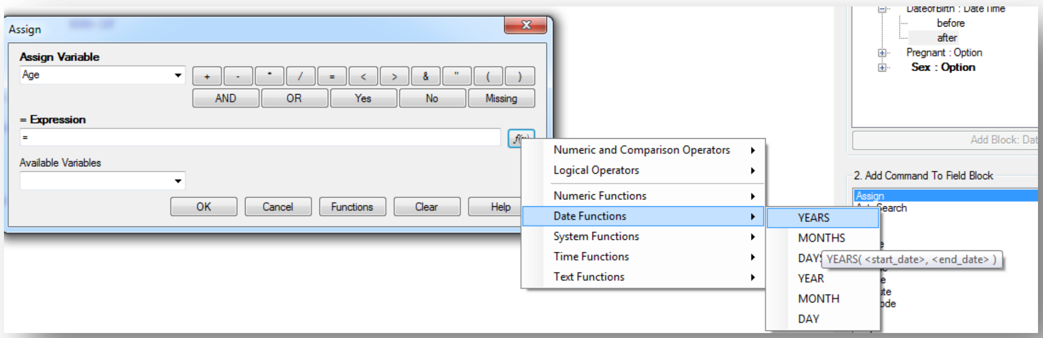
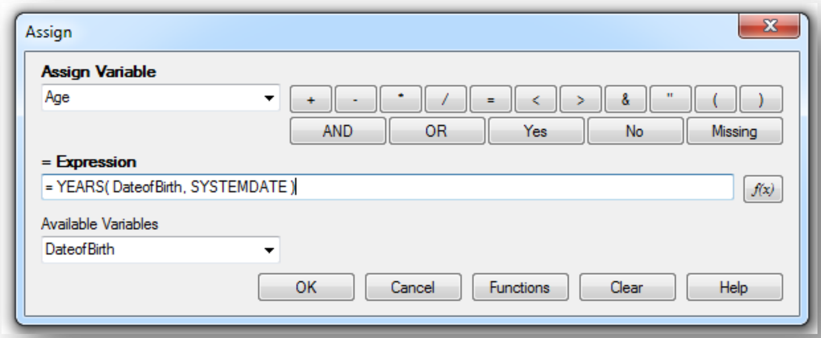


Figure 3.17: Assign Date Function options



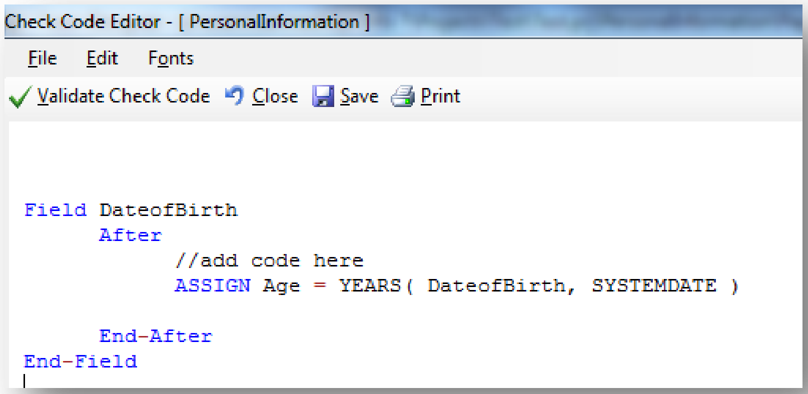
Assign Variable
Age

= Expression
= YEARS(DateofBirth, SYSTEMDATE)

Available Variables
DateofBirth

Figure 3.18: Assign dialog box

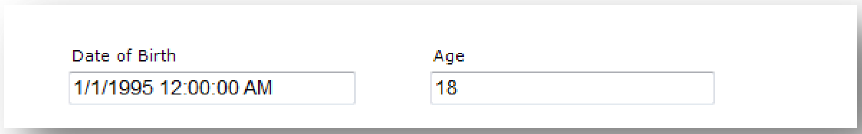
17. Click **OK**. Check Code will appear in the Check Code Editor.
18. Click the **Verify Check Code** button in the Check Code Editor.
19. Click the **Save** button in the Check Code Editor.


Check Code Editor - [PersonalInformation]
File Edit Fonts
Validate Check Code Close Save Print

```
Field DateofBirth
      After
            //add code here
            ASSIGN Age = YEARS( DateofBirth, SYSTEMDATE )

      End-After
End-Field
```

Figure 3.19: Assign Check Code Command

When **Date of Birth** is entered into the form, the **Age** field will automatically populate.



Date of Birth: 1/1/1995 12:00:00 AM

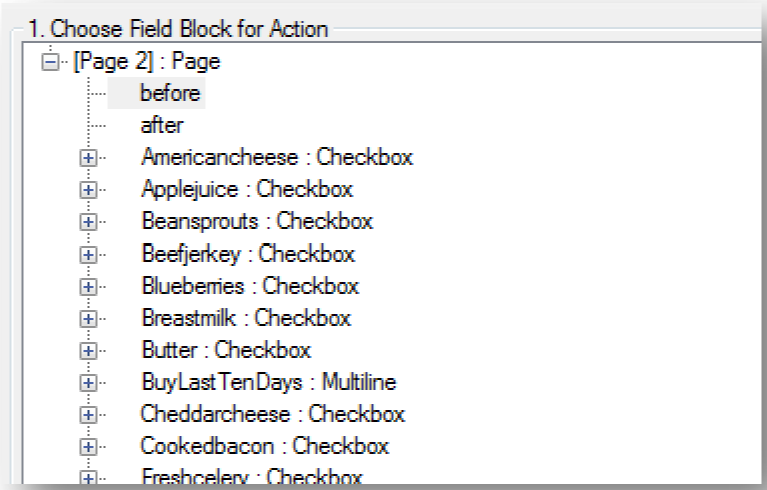
Age: 18

Figure 3.20: Assignment of Age in Enter Data

Interact with Users with the DIALOG command

The **DIALOG** command provides interaction with data entry personnel from within the program. Dialogs can display information, ask for and receive input, and offer lists to make choices. In the following example, the **DIALOG** command creates a reminder that all fields on page two of the survey must be completed.

1. From Form Designer, Open the **FoodHistory_NoCheckCode** form in the **EColi.prj**.
2. Click **Check Code** or select **Tools > Check Code Editor**. The Check Code Editor opens.
3. From the **Choose Field Block for Action** list box, select **Page 2**. The action should occur before the page is loaded.
4. Select **Before** from the Before or After Section.
5. Click the **Add Block** button.

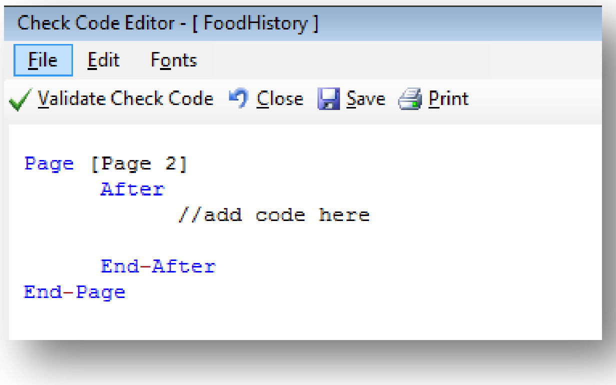


1. Choose Field Block for Action

```
[Page 2] : Page
    before
    after
    Americancheese : Checkbox
    Applejuice : Checkbox
    Beansprouts : Checkbox
    Beefjerkey : Checkbox
    Blueberries : Checkbox
    Breastmilk : Checkbox
    Butter : Checkbox
    BuyLastTenDays : Multiline
    Cheddarcheese : Checkbox
    Cookedbacon : Checkbox
    Freshcelery : Checkbox
```

Figure 3.21: Dialog Block for Action

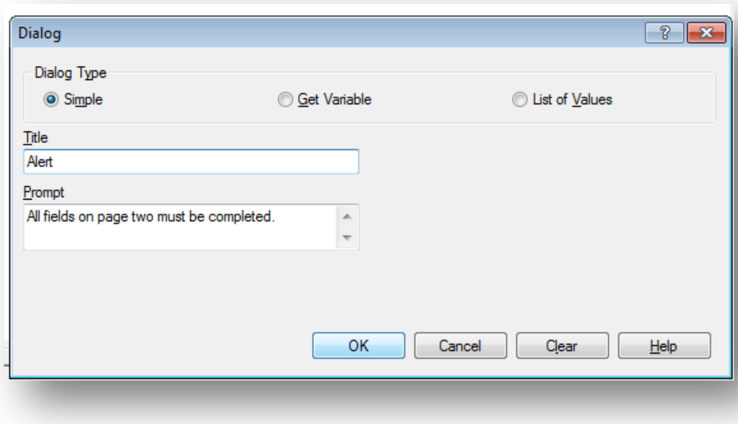
6. The code appears in the Check Code Editor.



```
Page [Page 2]
      After
            //add code here

      End-After
End-Page
```

Figure 3.22: Dialog Block Code

7. From the **Add Command to Field Block** list box, select **Dialog**. The **DIALOG** box opens.
8. Select **Simple** from the **Dialog Type** radio button options.
9. In the **Title** field, type ***Alert***.
10. In the **Prompt** field, type ***All fields on page two must be completed***.



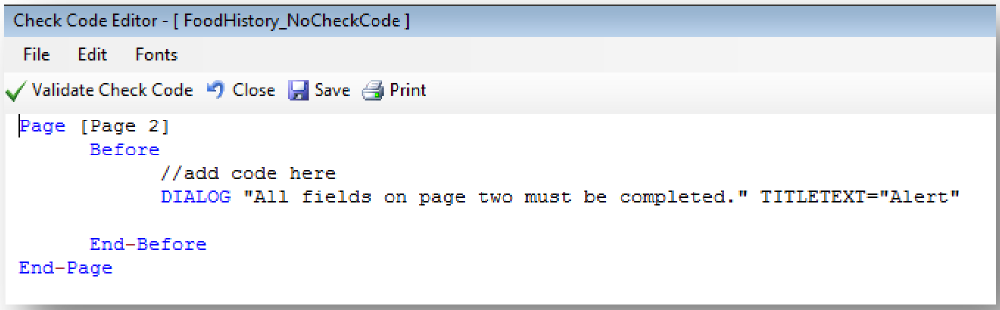
Dialog Type: [✓ Simple] Get Variable List of Values

Title: Alert

Prompt: All fields on page two must be completed.

Figure 3.23: Dialog command box

11. Click **OK**. The code appears in the Check Code Editor.


Check Code Editor - [FoodHistory_NoCheckCode]
File Edit Fonts
Validate Check Code Close Save Print

```
Page [Page 2]
      Before
            //add code here
            DIALOG "All fields on page two must be completed." TITLETEXT="Alert"

      End-Before
End-Page
```

Figure 3.24: Dialog Check Code Command

12. Click the **Verify Check Code** button from the Check Code Editor.
13. Click the **Save** button in the Check Code Editor.

Searching for Records – Using the AUTOSEARCH command

The Autosearch command automatically searches for an existing record that matches the entered values and notifies the user. A choice is then offered between editing the matching record or continuing with data entry in the new record. Duplicate records are detected and may be prevented with the AutoSearch command. For this example, we will incorporate the Autosearch command on the **CaseID** field.

1. From Form Designer, Open the **FoodHistory_NoCheckCode** form in the **EColi.prj**.
2. Click **Check Code** or select **Tools > Check Code Editor**. The Check Code Editor opens.
3. From the **Choose Field Block for Action section,** expand the node for the page where the **CaseID** field is located, which in this example is Page 1.
4. Expand the node for the field **CaseID**.
5. Double click on the **after** event.
6. A block of code for the **CaseID** field is created and displayed in the Check Code Editor.

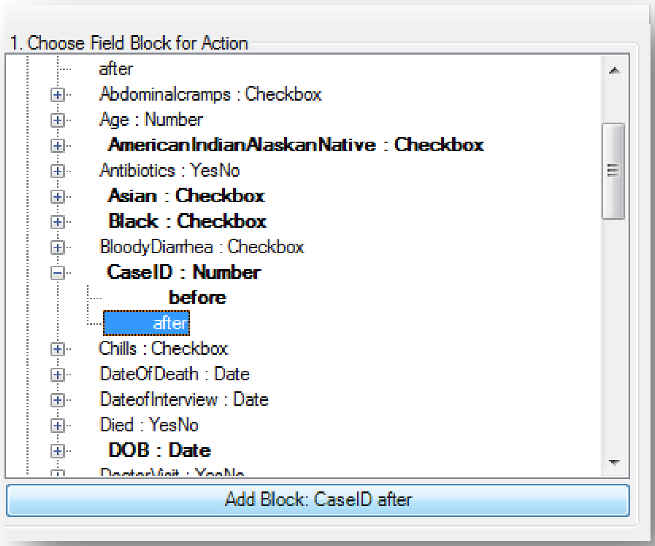
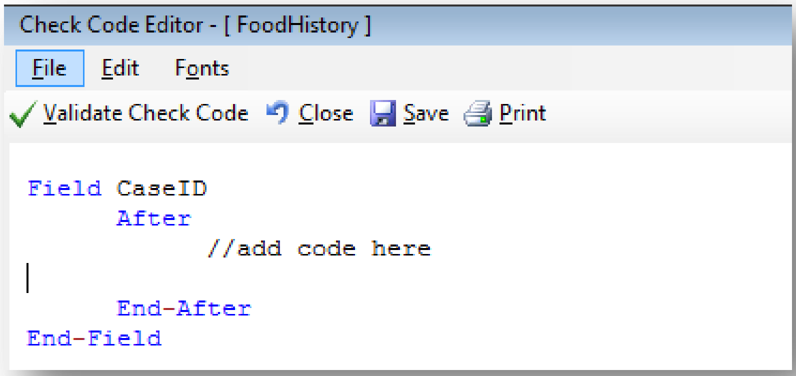


Figure 3.25: AutoSearch Block for Action

7. The code appears in the Check Code Editor.



```
Field CaseID
      After
            //add code here

      End-After
End-Field
```

Figure 3.26: AutoSearch Block Code

8. From the **Add Command to Field Block** list box, click **Autosearch**. The Autosearch window opens.
9. Select the variable(s) to be searched during data entry. In this example, select **CaseID**.

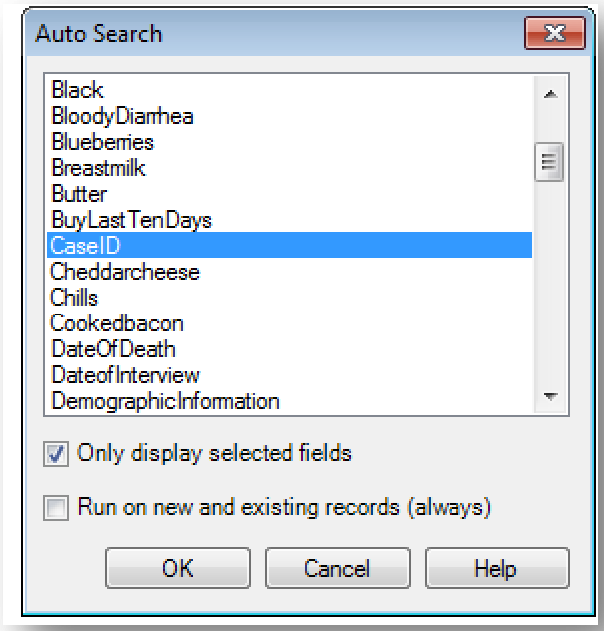
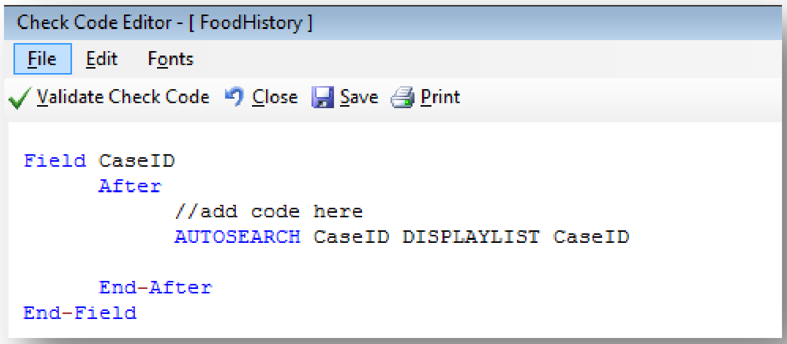


Figure 3.27: Auto Search dialog box

10. Click **OK**. The code appears in the Check Code Editor window.



```
Field CaseID
      After
            //add code here
            AUTOSEARCH CaseID DISPLAYLIST CaseID

      End-After
End-Field
```

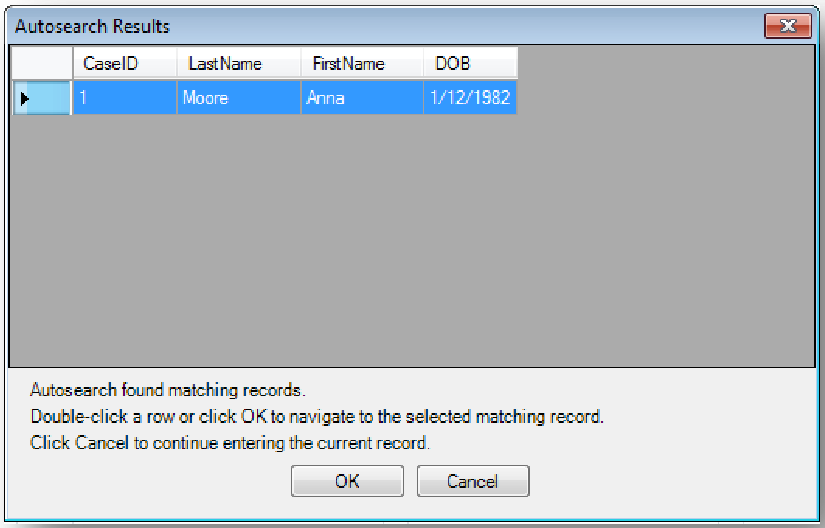
Figure 3.28: AutoSearch Check Code Command

11. Click the **Verify Check Code** button from the Check Code Editor.
12. Click the **Save** button from the Check Code Editor.
13. .Click **Close** to return to the form.

When a duplicate record is entered in the Enter Data tool, the **Autosearch** dialog box opens with all the matching records listed. To view the potential duplicate record, double-click the arrow that appears next to the record. The field where the potential duplicate was entered is cleared. Alternatively, click **Cancel** to remain on the current record and accept the duplicate value. To display other variables when a matching record is found, add the variable names after the **DISPLAYLIST** parameter (i.e. Last Name, First Name and Date of Birth as shown below).

```
Field CaseID
      After
      //add code here
            AUTOSEARCH CaseId DISPLAYLIST CaseID LastName FirstName DOB
      End-After
End-Field
```

In the check code above, the autosearch will be done on the field **CaseID** and if any matching records are detected, the fields **CaseID, LastName, FirstName** and **DOB** will be displayed on the grid.



CaseID	LastName	FirstName	DOB
1	Moore	Anna	1/12/1982

Autosearch found matching records.
Double-click a row or click OK to navigate to the selected matching record.
Click Cancel to continue entering the current record.

Figure 3.29: Autosearch Results

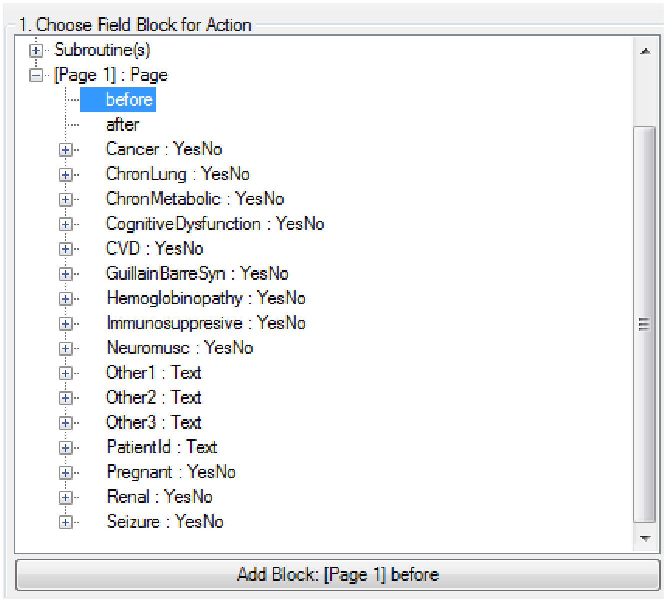
Note: For more information on using Autosearch, please see the Autosearch topic in the Command Reference.

Copy the Value of a Field from a Main Form to a Related Form

When users have developed a relational database setup using Epi Info™ 7, it might be appropriate to transfer values entered in the parent form (i.e. core demographics) into the child form (i.e. Visits information). In order to accomplish this process, Check Code must be created for a value from a field in the main form to appear in a related form (i.e., there may be a Case ID Number or Patient's Name that needs to be visible in the parent and child forms).

The following instructions assume the parent and child forms already exist. Let's assume that the Parent form is called **Surveillance** while the Child form is called **Hepatitis.** The field to be copied needs to exist in the parent form or be created in the parent form prior to the incorporation of the check code in the child form (i.e. Last Name in parent form will be passed to Last Name in child form). Let's make the assumption that the name of the field on the Parent form whose value will be copied to the Child form is called **PatientId**.

1. From Form Designer, open your project and click on the child form name from the Project Explorer tree.
2. Create a new field. The new field must be the same field type as the field being copied from the parent form. For this example, use **PatientId**.
3. Select the **Read Only** option.
4. Click **OK**. The new field appears in the form. This is where the value from the parent form will be assigned and displayed during data entry on the child form.
5. Click **Check Code** or select **Tools > Check Code Editor**. The Check Code Editor opens.
6. From the **Choose Field Block for Action** list box, select the page corresponding to where the PatientId field was placed. Let's make the assumption that this field was created on Page 1. For the Page, select the **before** from the Before or After Section.
7. Click the **Add Block** button.


1. Choose Field Block for Action

Subroutine(s)
[Page 1] : Page
before
after
Cancer : YesNo
ChronLung : YesNo
ChronMetabolic : YesNo
CognitiveDysfunction : YesNo
CVD : YesNo
GuillainBarreSyn : YesNo
Hemoglobinopathy : YesNo
Immunosuppresive : YesNo
Neuromusc : YesNo
Other1 : Text
Other2 : Text
Other3 : Text
PatientId : Text
Pregnant : YesNo
Renal : YesNo
Seizure : YesNo

Figure 3.30: Choose Field Block for Action

8. From the **Add Command to Field Block list** box, click **Assign**. The **ASSIGN** dialog box appears.
9. From the **Assign Variable** drop-down list, select the new variable, **PatientId**.

10. In the = Expression area, type the **field name** from the parent form, in this case **PatientId**. This field name must be prefixed by the parent form's name followed by a period in the Assign expression. We will need to make that modification once the command is written into the Program Editor.:

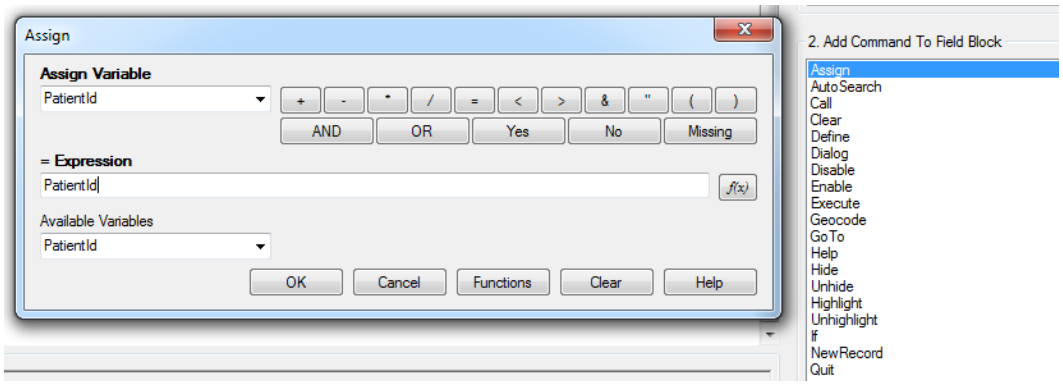
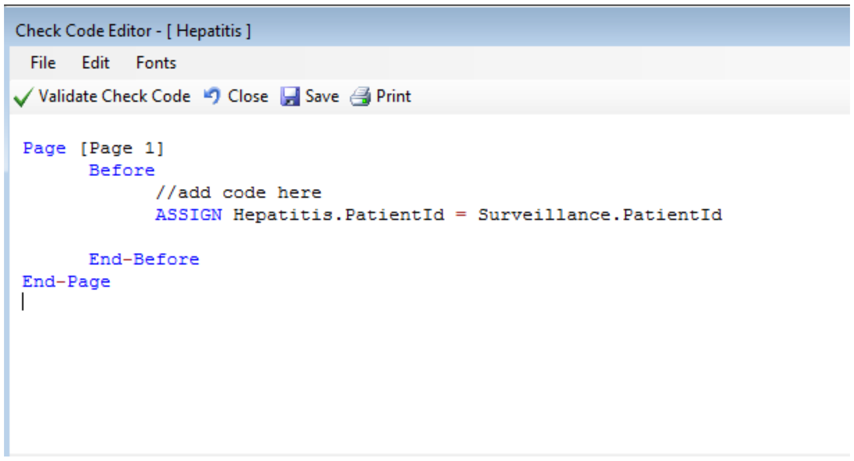


Figure 3.31: Copy Value- Assign dialog box

11. Click **OK**.
12. The code appears in the Check Code Editor. If the child form already has a field with the same name as the one being copied from the parent form, it is important to distinguish the parent's field name. This field name must be prefixed by the parent form's name followed by a period in the Assign expression. In this example, Hepatitis is the name of the child form while Surveillance is the name of the parent form. If the child form does not have a field with the same name as the one being copied from the parent form, it is sufficient to indicate just the field name. Therefore, for this example, this modification will be required to the syntax since the field is called the same on both forms. Once completed, your syntax should look like the one on Figure 3.32.


Check Code Editor - [Hepatitis]
File Edit Fonts
Validate Check Code Close Save Print

```
Page [Page 1]
      Before
            //add code here
            ASSIGN Hepatitis.PatientId = Surveillance.PatientId

      End-Before
End-Page
```

Figure 3.32: Copy Value Check Code Command

13. Click the **Validate Check Code** button and, if needed, correct any issues.
14. Click **Save.**

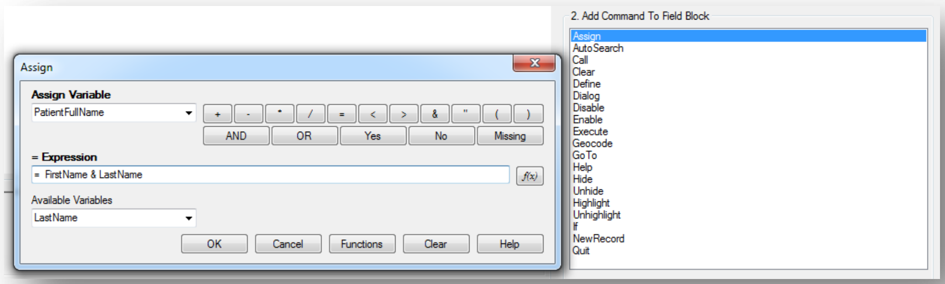
Concatenate Fields

When writing check code for concatenating fields, syntax will be executed for new records entered. The code will not go back and populate previously entered data. If previously entered data need to be concatenated, use concatenation commands from the Classic Analysis section of the manual.

Concatenate Fields with the Ampersand '&' Operator

This example illustrates how to join data from two fields and assign it into a third field using the '&' operator. In this example, **Patient Full Name** will be assigned to the concatenation of **First Name** and **Last Name**.

1. In Form Designer, open your form.
2. Click **Check Code**. The Check Code Editor opens.
3. From the **Choose Field Block for Action** list box, select **LastName**.
4. Select **after** from the Before or After Section.
5. Click the **Add Block** button.
6. From the **Add Command to Field Block list** box, click **Assign**. The **ASSIGN** dialog opens.
7. From the **Assign Variable** drop-down list, select the field to contain the concatenated value.
8. Create the = Expression using the '&' operator. In this example, **ASSIGN PatientFullName = FirstName & LastName**.



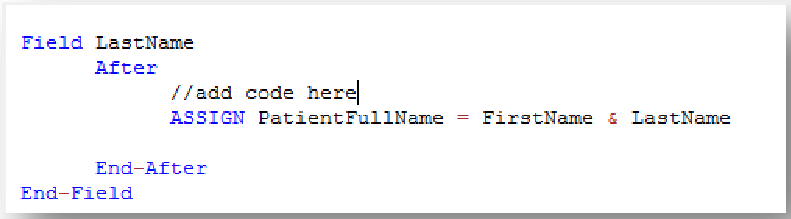
Assign Variable: PatientFullName

= Expression: = FirstName & LastName

Available Variables: LastName

Figure 3.33: Concatenate- Assign dialog box

9. Click **OK**. Check Code appears in the Check Code Editor.

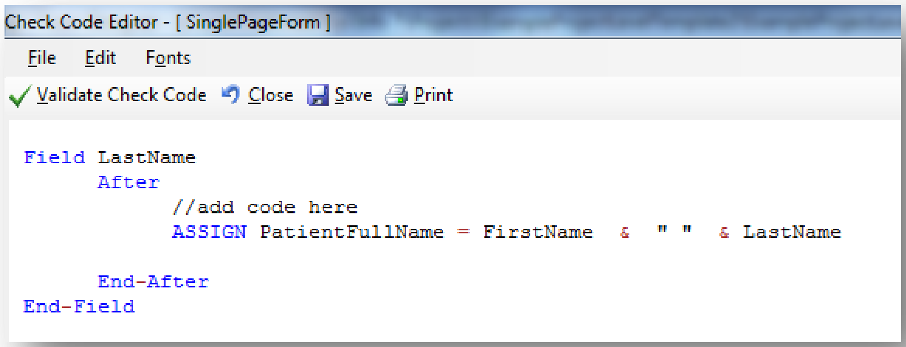


```
Field LastName
      After
            //add code here
            ASSIGN PatientFullName = FirstName & LastName

      End-After
End-Field
```

Figure 3.34: Concatenate No Space Check Code Command

10. Click the **Validate Check Code** button and correct any issues.
11. Click **Save.**

If in the Enter tool, **Carl** was entered for FirstName and **Gao** was entered for LastName, the result of PatientFullName would be **CarlGao**. There are no spaces between the names. To add a space between the names, simply modify the ASSIGN statement by adding a blank space in quotes between the first and last names as in the following statement:


Check Code Editor - [SinglePageForm]
File Edit Fonts
Validate Check Code Close Save Print

```
Field LastName
      After
            //add code here
            ASSIGN PatientFullName = FirstName  &  " "  & LastName

      End-After
End-Field
```

Figure 3.35: Concatenate Include Space Check Code Command

Concatenate Fields with the Substring Function

This example illustrates how to join parts of two variables to create a unique text ID. In this example, you will create a Patient ID made up of parts of the patient's last and first name. The ampersand (&) operator is used to join the two parts together.

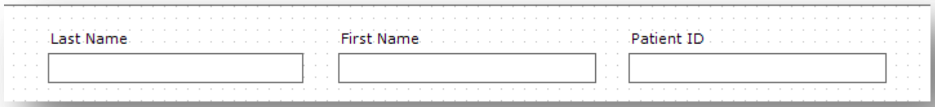


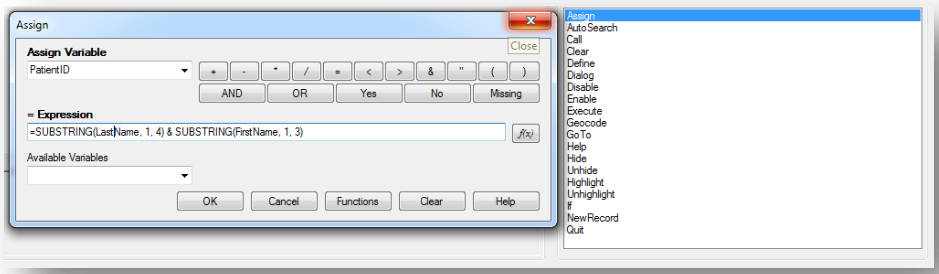
Figure 3.36: Enter Form - Name

1. From the Form Designer, click **Check Code**. The Check Code Editor opens.

2. From the **Choose Field Block for Action** list box, select **FirstName**.
3. Select **after** from the Before or After Section.
4. Click the **Add Block** button.
5. From the **Add Command to Field Block** list box click **Assign**. The **ASSIGN** dialog box opens.
6. From the **Assign Variable** drop-down list, select the field to contain the concatenated value. In this example, select **PatientID**.
7. Create the =Expression using the SUBSTRING syntax.

- SUBSTRING(<variable>, position #, #characters)
- <variable> is the field or variable
- position # is the position of the first character to be extracted from the variable
- #characters is the number of characters to extract

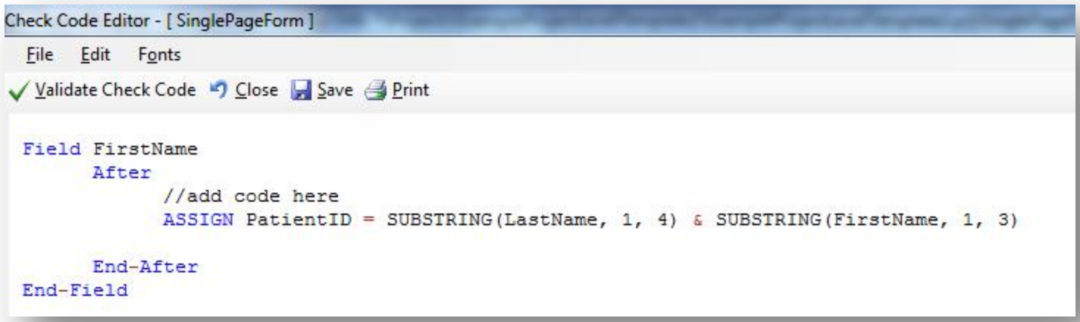
In this example, the **PatientID** variable contains a combination of the first position and four characters of the last name plus the first position and three characters of the first name.


Assign Variable: PatientID

= Expression: =SUBSTRING(LastName, 1, 4) & SUBSTRING(FirstName, 1, 3)

Figure 3.37: Concatenate with Substring Assign dialog box

8. Click **OK**. The code appears in the Check Code Editor window.



```
Field FirstName
      After
            //add code here
            ASSIGN PatientID = SUBSTRING(LastName, 1, 4) & SUBSTRING(FirstName, 1, 3)

      End-After
End-Field
```

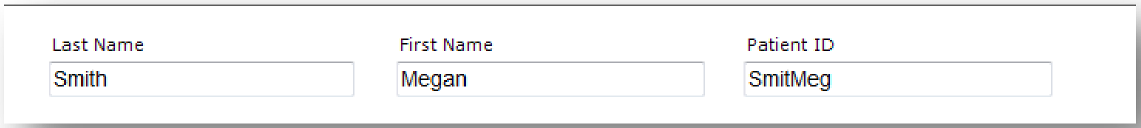
Figure 3.38: Concatenate with Substring Check Code Command

9. Click the **Validate Check Code** button and correct any issues.

10. Click the **Save** button**.**

The example functions as such:

- Last Name: Smith
- First Name: Megan
- Patient ID: SmitMeg
- The **Patient ID** field being calculated is Read Only.



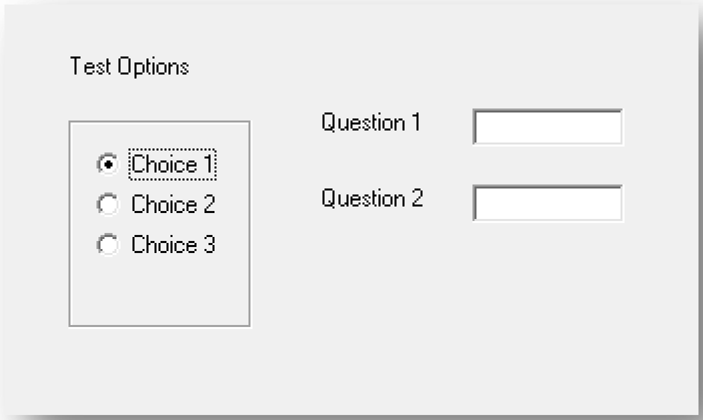
Last Name	First Name	Patient ID
Smith	Megan	SmitMeg

Figure 3.39: Concatenate with Substring Enter Form

Create Check Code for Option Box Fields

Check Code can be added to option box variables. Code can be added to any line/choice present in the option boxes. Use the Check Code Editor to create complex Check Code for option box variables.

For this example, the check code created uses the **GOTO** command. Check Code was used to create the following scenario. If the answer to *TestOptions* is Choice 1, the cursor will jump to *Question 2*. If the answer to *TestOptions* is *Choice 2* or *Choice 3*, the cursor will jump to *Question 1*. Check the tab order before creating the Check Code to ensure that the tab order is correct.



Test Options

[✓ Choice 1] Choice 2 Choice 3

Question 1

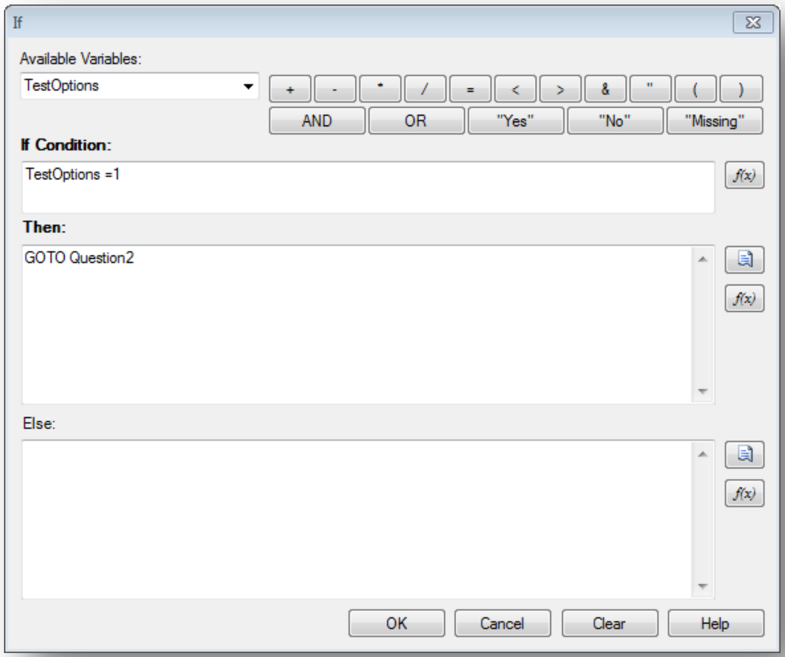
Question 2

Figure 3.40: Text Options box

1. Create an Option Box in a form. Note the name of the variable.

- The variable is named TestOptions.
- Each text line that represents a choice in the form represents a numeric position in the Check Code Editor.
- For example, there are three lines/choices that can be made in the variable *TestOptions*. In the Check Code Editor the choices are numeric, choice 1 = position 0, choice 2 = position 1, and choice 3 = position 2.

2. Open the **Check Code Editor**.
3. From the Choose Field Block for Action list box, select **TestOptions**.
4. Select **after** from the Before or After Section.
5. Click the **Add Block** button.
6. From the **Add Command to Field Block** list box, click **If**. The **IF** dialog box opens.
7. From the **If Condition** field, type **Test Options = 1**.
8. Remember that the number 1 in this instance represents a text value called *Choice 2*.
9. Click the **Code Snippet** button in the **Then** section. A list of available commands appears.
10. From the command list, select **GoTo**. The **GOTO** dialog box opens.
11. Select **Question2**.
12. Click **OK**.


Available Variables: TestOptions

If Condition:
```
TestOptions =1
```
Then:
```
GOTO Question2
```
Else:

Figure 3.41: If dialog box

13. Click **Validate Check Code** and correct any issues.
14. Click **Save.**

Delete a Line of Code from the Check Code Editor

To delete a line(s) of check code from the Check Code Editor, you can complete the following steps:

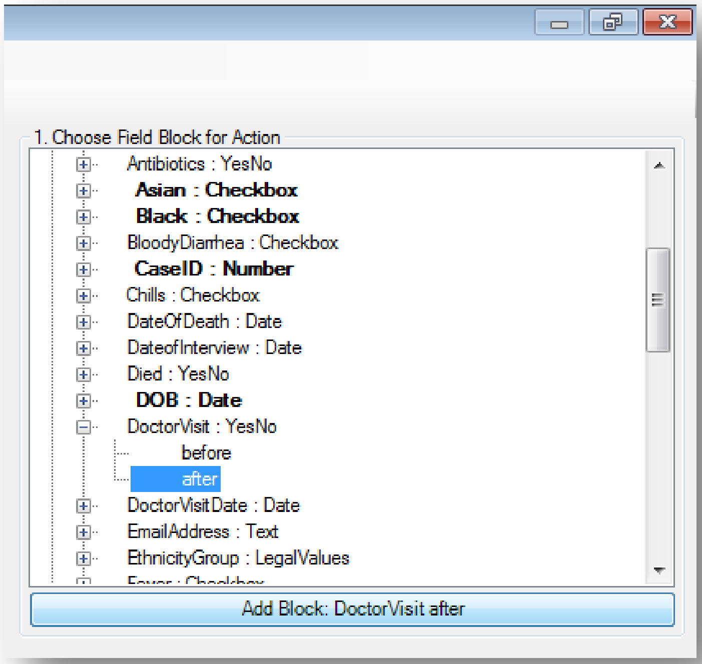
1. Highlight the line(s) of code/text that you desire to delete**.**
2. Press the **Delete** key on your keyboard.
3. Once done, from the Check Code Editor toolbar, click **Save**.

Note: Be sure of all deletions made. No confirmation prompt or undo button will appear prior to deletion.

Disable

If there isn't a need to capture information for a particular field, then it can be disabled. Disable is usually used with the IF, THEN, ELSE conditions. In this example, the field DoctorVisitDate will be disabled if the response to the DoctorVisit is *No*.

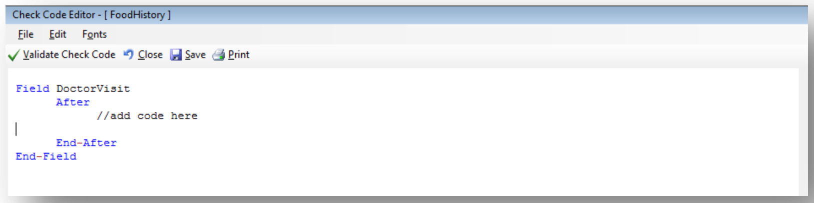
1. Open the **FoodHistory_NoCheckCode** form in the **EColi.prj.**
2. Click **Check Code**. The Check Code Editor opens.
3. Select the **DoctorVisit** from the **Choose Field Block for Action** list box.
4. Select **after** from the Before or After Section.
5. Click the **Add Block** button.


1. Choose Field Block for Action

Antibiotics : YesNo
Asian : Checkbox
Black : Checkbox
BloodyDiarrhea : Checkbox
CaseID : Number
Chills : Checkbox
DateOfDeath : Date
DateofInterview : Date
Died : YesNo
DOB : Date
DoctorVisit : YesNo
before
after
DoctorVisitDate : Date
EmailAddress : Text
EthnicityGroup : LegalValues

Figure 3.42: Disable Block for Action

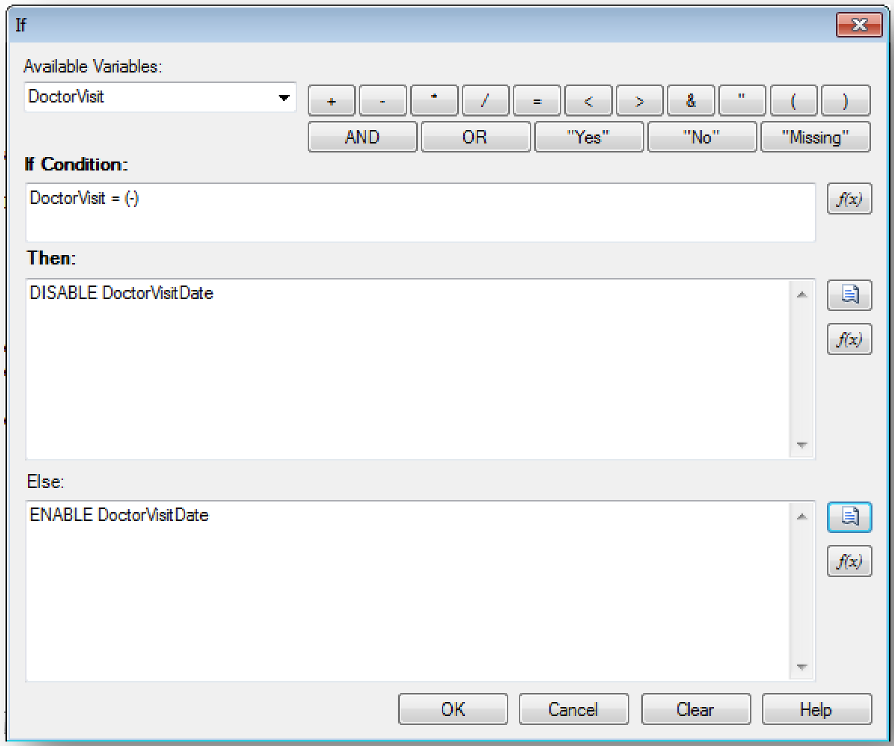
6. The code block appears in the Check Code Editor.


Check Code Editor - [FoodHistory]
File Edit Fonts
Validate Check Code Close Save Print

```
Field DoctorVisit
      After
            //add code here

      End-After
End-Field
```

Figure 3.43: Disable Block Command

7. Click **IF** from the **Add Command to Field Block** list box. The **IF** dialog box opens.
8. From the **Available Variables** drop-down list, select the **field** to contain the action. For this example, select **DoctorVisit**. The selected variable appears in the **If Condition** field.
9. From the Operators, click **=**.
10. From the Operators, click **No**. The **If Condition** field will read DoctorVisit=(-).
11. Click the **Code Snippet** button in the **Then** section. A list of available commands appears.
12. From the command list, select **DISABLE**. The **DISABLE** dialog box opens.
13. Select the **field** to be disabled based on a **No** answer from the list of variables. For this example, select **DoctorVisitDate.**
14. Click **OK** to return to the **IF** dialog box.
15. Click the **Code Snippet** button in the **Else** section. A list of available commands appears.
16. From the command list, select **Enable**. The **Enable** dialog box opens.
17. Select the field to enable based on a **Yes** answer from the list of variables. For this example, select **DoctorVisitDate.**
18. In the **Enable** dialog box, click **OK** to return to the **If** dialog box.



Available Variables: DoctorVisit

If Condition:

```
DoctorVisit = (-)
```

Then:

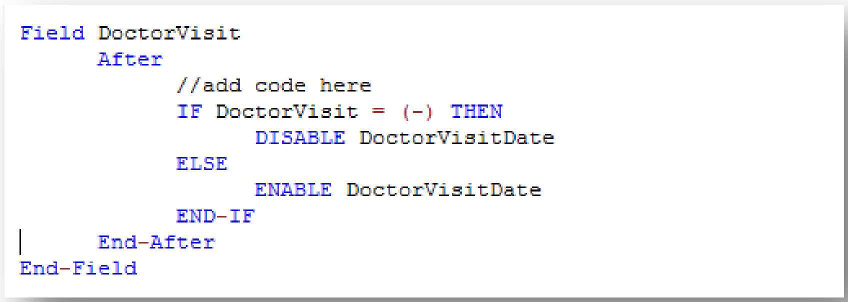
```
DISABLE DoctorVisitDate
```

Else:

```
ENABLE DoctorVisitDate
```

Figure 3.44: Disable- If dialog box

19. Click **OK**. The code appears in the Check Code Editor. The example code appears as:



```
Field DoctorVisit
      After
            //add code here
            IF DoctorVisit = (-) THEN
                  DISABLE DoctorVisitDate
            ELSE
                  ENABLE DoctorVisitDate
            END-IF
      End-After
End-Field
```

Figure 3.45: Disable Check Code Command

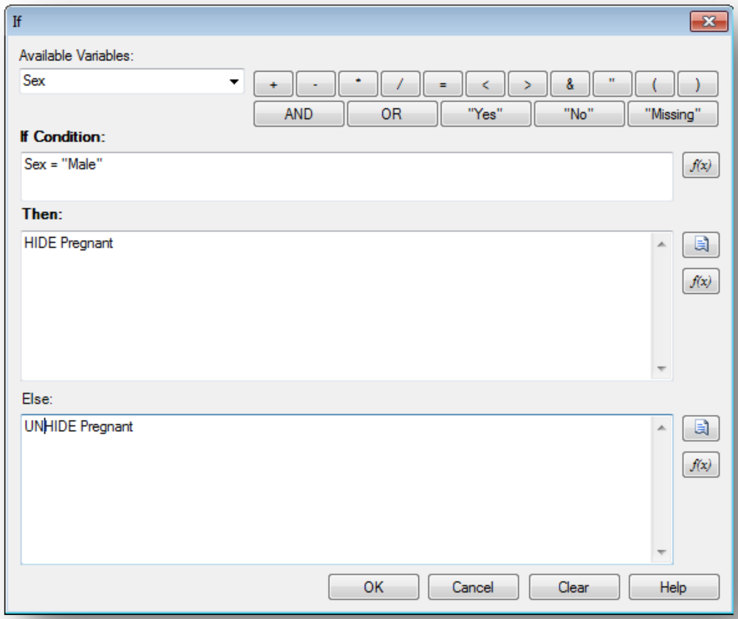
20. Click Verify Check Code button.
21. Click **Save**.
22. Click **Close** to return to the form.

To test the Disable, open the form in the Enter Data tool. When **No** is entered for **DoctorVisit**, **DoctorVisitDate** should become disabled.

Hide

A field may be hidden if there is not a need to capture information for a particular field or if it does not apply based on previously answered questions. The field on the form would be hidden from the form. In the following example, the Pregnant option box will be hidden based on the users Sex; Female or Male.

1. Click **Check Code**. The Check Code Editor opens.
2. Select **Sex** from the **Choose Field Block for Action** list box.
3. Select **after** from the Before or After Section.
4. Click the **Add Block** button. This creates code to run after data is entered and accepted.
5. The code appears in the **Check Code Editor**.
6. Click **IF** from the **Add Command to Field Block list** box. The **IF** dialog box opens.
7. From the **Available Variables** drop-down list, select the **field** to contain the action. For this example, select **Sex**. The selected variable appears in the **If Condition** field.
8. From the Operators, click **=**.
9. From the **If Condition** field, type **Sex = "Male"**. Remember that Sex is a text field and the value must be enclosed in quotes.
10. Click the **Code Snippet** button in the **Then** section. A list of available commands appears.
11. From the command list, select **Hide**. The **Hide** dialog box opens.
12. Select **Pregnant**.
13. Click **OK.**



Available Variables: Sex

If Condition:

Sex = "Male"

Then:

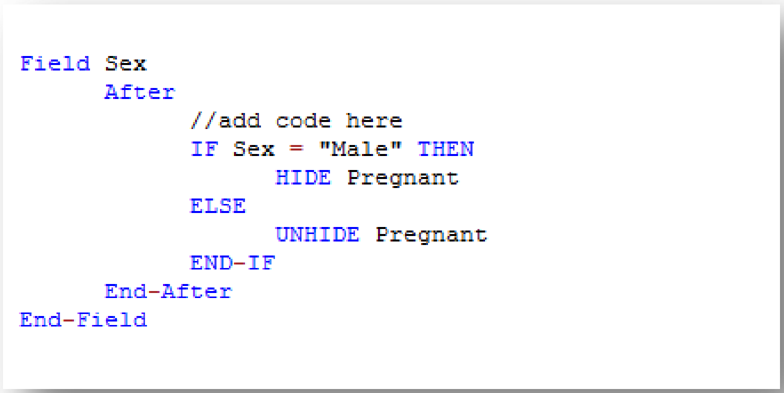
HIDE Pregnant

Else:

UNHIDE Pregnant

Figure 3.46: Hide- If dialog box

14. Click the **Code Snippet** button in the **Else** section. A list of available commands appears.
15. From the command list, select **UNHIDE**. The **Unhide** dialog box opens.
16. Select the field to enable based on a **Yes** answer from the list of variables. For this example, select **Pregnant.**
17. In the **Unhide** dialog box, click **OK** to return to the **If** dialog box.
18. Click **OK**.
19. Click **OK**. The code appears in the Check Code Editor.



```
Field Sex
      After
            //add code here
            IF Sex = "Male" THEN
                  HIDE Pregnant
            ELSE
                  UNHIDE Pregnant
            END-IF
      End-After
End-Field
```

Figure 3.47: Hide Check Code Command

20. Click the **Save** button.

21. Click **Close** to return to the form.

Note: The Unhide command will display any hidden fields. You can select this command in the Add Command to Field Block section of the Check Code Editor. It is recommended to also incorporate a CLEAR command with the HIDE command in order to set null any information previously entered into the Pregnant field.

Geocode

The Geocode command uses the text entered into Address to retrieve and populate Latitude and Longitude coordinates into your form.

1. From Form Designer, Open the **FoodHistory_NoCheckCode** form in the **EColi.prj**.
2. Click **Check Code**. The Check Code Editor opens.
3. Select **GetCoordinates** from the **Choose Field Block for Action** list box.
4. Select **Click**.
5. Click the **Add Block** button. This creates code to run after the **Get Coordinates** command button is clicked.

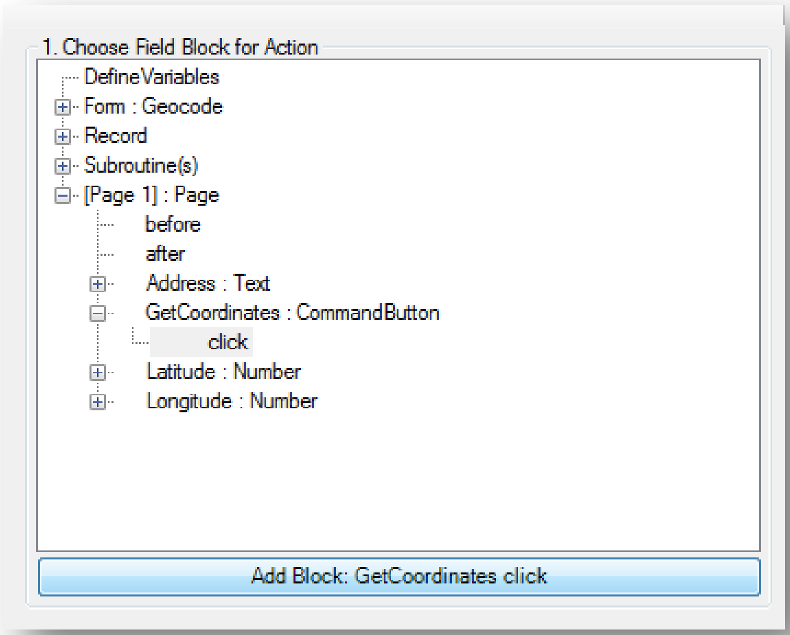


Figure 3.48: Geocode Block for Action

6. The code appears in the Check Code Editor.
7. Select **Geocode** from the **Add Command to Field Block** list box. The **Geocode** dialog box opens.
8. From the **Address** drop-down list, select **Address**.

9. From the **Latitude** drop-down list, select **Latitude**.
10. From the **Longitude** drop-down list, select **Longitude**.

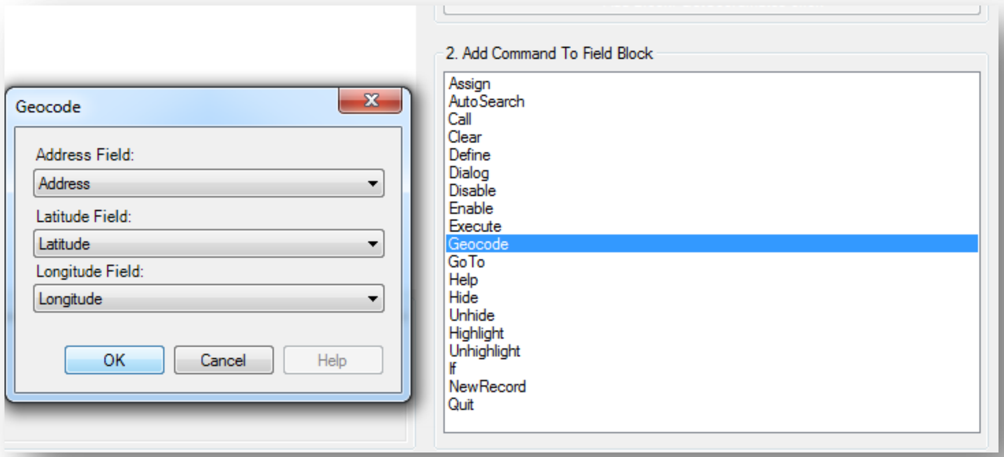
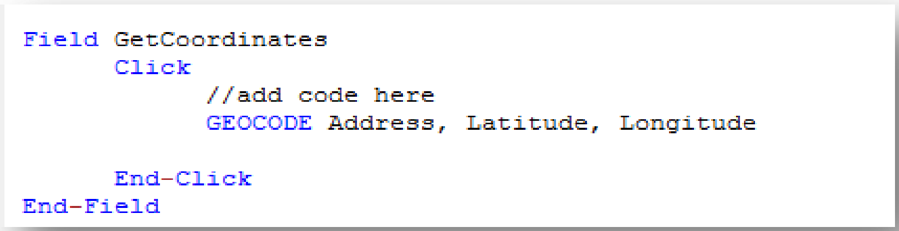


Figure 3.49: Geocode dialog box

11. Click **OK.**
12. Click **OK**. The code appears in the Check Code Editor.

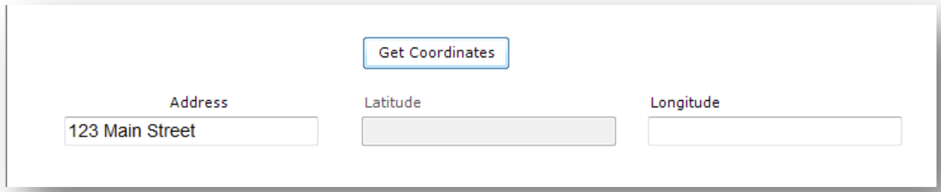


```
Field GetCoordinates
      Click
            //add code here
            GEOCODE Address, Latitude, Longitude

      End-Click
End-Field
```

Figure 3.50: Geocode Check Code Command

13. Click the **Save** button.
14. Click **Close** to return to the form.

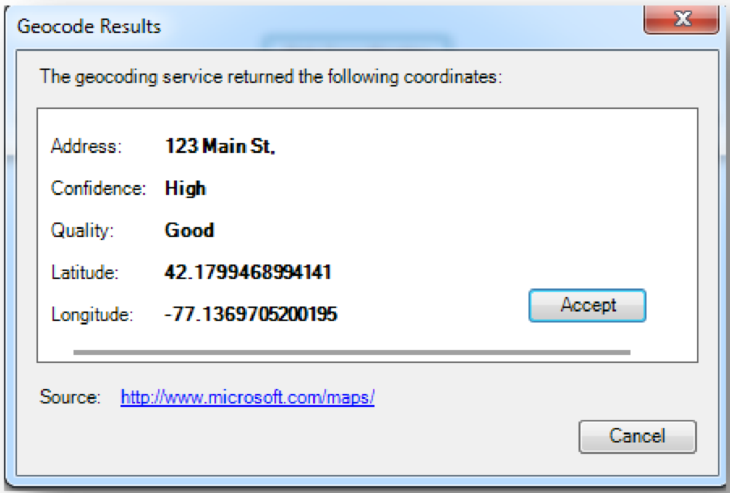
When the **Get Coordinates** command button is clicked after an address is entered into your form in the Enter Data tool, the **Geocode Results** dialog will appear.



Address	Latitude	Longitude
123 Main Street		

Figure 3.51: Geocode Enter Data Before

The **Geocode Results** dialog box contains, the Address entered into the form along with the Latitude and Longitude coordinates of that address. The confidence and quality of the geocoding service coordinates are also displayed.



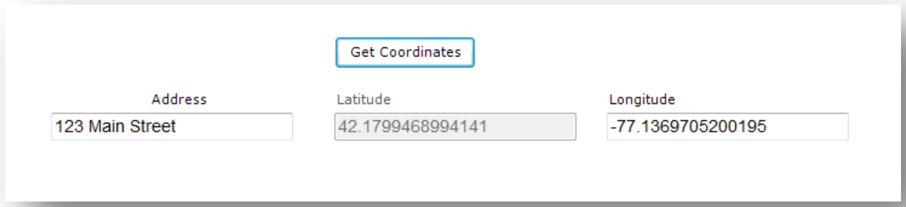
The geocoding service returned the following coordinates:

Address:	**123 Main St.**
Confidence:	**High**
Quality:	**Good**
Latitude:	**42.1799468994141**
Longitude:	**-77.1369705200195**

Source: http://www.microsoft.com/maps/

Figure 3.52: Geocode Results

After clicking **Accept** in the **Geocode Results** dialog box, the coordinates provided by the geocoding service are copied into the **Latitude** and **Longitude** fields in your form.



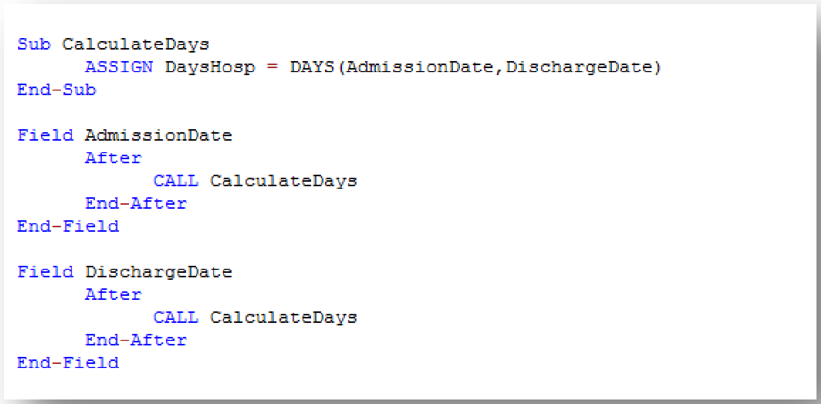
Address	Latitude	Longitude
123 Main Street	42.1799468994141	-77.1369705200195

Figure 3.53: Geocode Enter Data After

Additional Check Code Commands

Call

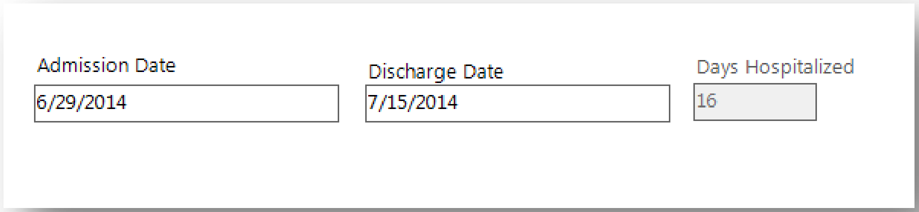
This command redirects to another command block in check code and returns after it has been executed. This command is commonly used with subroutines. Subroutines act as a unit of common check code, which is usually dependent on two variables. The benefit of using subroutines is that it allows maintenance of check code in one common location. Here is an example of a subroutine:



```
Sub CalculateDays
      ASSIGN DaysHosp = DAYS(AdmissionDate,DischargeDate)
End-Sub

Field AdmissionDate
      After
            CALL CalculateDays
      End-After
End-Field

Field DischargeDate
      After
            CALL CalculateDays
      End-After
End-Field
```

Figure 3.54: Check code syntax for Subroutines

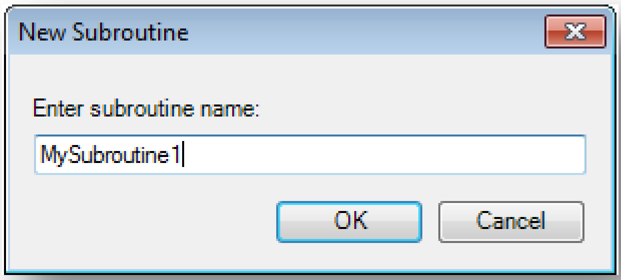
In the example above, the calculation of days hospitalized would have been required to be placed in the AFTER event of the Admission Date and Discharge Date fields. However, through the usage of subroutines, users can update and maintain check code only in one location. By using the CALL command, the block of check code will execute when data are entered or updated in either field.



Admission Date	Discharge Date	Days Hospitalized
6/29/2014	7/15/2014	16

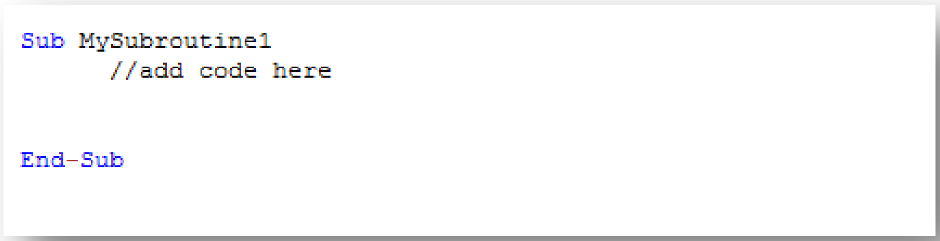
To create a subroutine, complete the following steps:

1. From Form Designer, Open your form.
2. Click **Check Code**. The Check Code Editor opens.
3. From the **Choose Field Block for Action** list box, expand the Subroutine item.
4. Double click on the **Add new item.** The *New Subroutine* window opens.
5. Assign a name to your subroutine (i.e. MySubroutine1).



1.

6. Click **OK**.

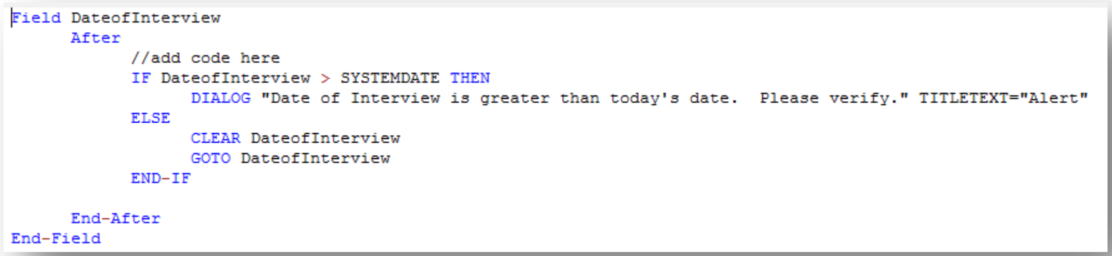


```
Sub MySubroutine1
      //add code here


End-Sub
```

A block of code for the subroutine called MySubroutine1 is created and displayed in the Check Code Editor. At this point, you can establish the check code commands to be incorporated into the subroutine. Make sure to Verify your code and Save once done.

Clear

CLEAR sets a field to a missing value, as though the field had been left blank. For example, it is useful to clear a previous entry in a field after an error has been detected. The CLEAR command can be followed by a GOTO command in order to place the cursor back into the field for further entry after an error. In the example below, the Date of Interview field is cleared after the user is notified that the date value entered is greater than today's date.



```
Field DateofInterview
      After
            //add code here
            IF DateofInterview > SYSTEMDATE THEN
                  DIALOG "Date of Interview is greater than today's date.  Please verify." TITLETEXT="Alert"
            ELSE
                  CLEAR DateofInterview
                  GOTO DateofInterview
            END-IF

      End-After
End-Field
```

Figure 3.55: Check code syntax for CLEAR command

Define

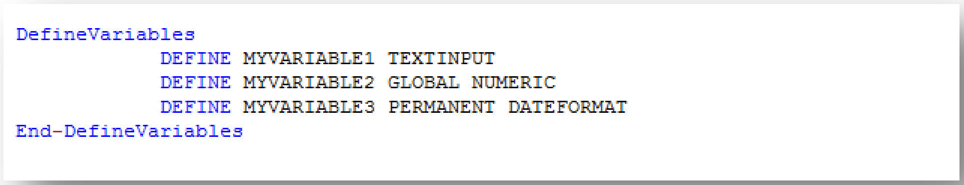
This command creates a new variable. In Check Code, all user-defined variables are saved in the DEFINEVARIABLES section.

The proper syntax is:

DEFINE <variable name> {<scope>} {<field type indicator>}

- **<variable name>** represents the name of the variable to be created. <variable> cannot be a reserved word. For a list of reserved words, see the List of Reserved Words section of the User's Manual.

- **<scope>** is optional and is the level of visibility and availability of the new variable. This parameter must be one of the reserved words: STANDARD, GLOBAL, or PERMANENT. If omitted, STANDARD is assumed and a type indicator cannot be used.
- **<field type indicator>** is the data type of the new variable and must be one of the following reserved words: NUMERIC, TEXTINPUT, YN, DATEFORMAT, TIMEFORMAT and DLLOBJECT. If omitted, the variable type will be inferred based on the data type of the first value assigned to the variable. Thereafter, the variable type cannot be changed. An attempt to assign data of a different type to the variable will result in an error.



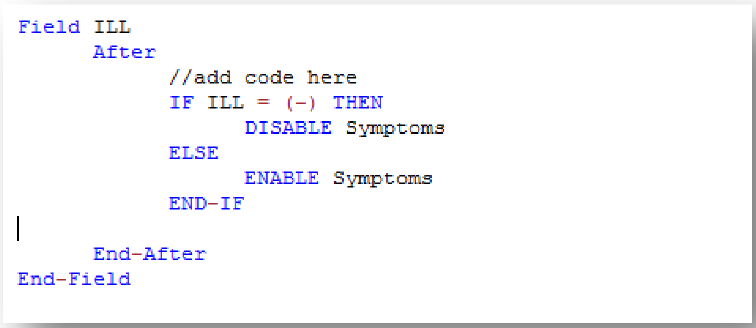
```
DefineVariables
            DEFINE MYVARIABLE1 TEXTINPUT
            DEFINE MYVARIABLE2 GLOBAL NUMERIC
            DEFINE MYVARIABLE3 PERMANENT DATEFORMAT
End-DefineVariables
```

Figure 3.56: Check code syntax for DEFINE command

Below is a description for the optional SCOPE parameter used with the DEFINE command.

- **STANDARD variables** retain their value only within the current record and are reset when a new record is loaded. Standard variables are used as temporary variables behaving like other fields in the database.
- **GLOBAL variables** retain values across related forms and when a new form is opened by the program, but are removed when the Enter program is closed. Global variables persist for the duration of program execution.
- **PERMANENT variables** are stored in the **EpiInfo.Config.xml** file and retain any value assigned until the value is changed by another assignment or the variable is undefined. Permanent variables are shared among Epi Info modules (Enter, Classic Analysis, etc.) and persist even if the computer is shut down.

Enable/Disable These two commands usually work in conjunction. The DISABLE command disallows data entry into a field while the ENABLE command allows data entry into a previously disabled field.



```
Field ILL
      After
            //add code here
            IF ILL = (-) THEN
                  DISABLE Symptoms
            ELSE
                  ENABLE Symptoms
            END-IF

      End-After
End-Field
```

Figure 3.57: Check code syntax for ENABLE/DISABLE command

Execute Use to execute a Windows program.

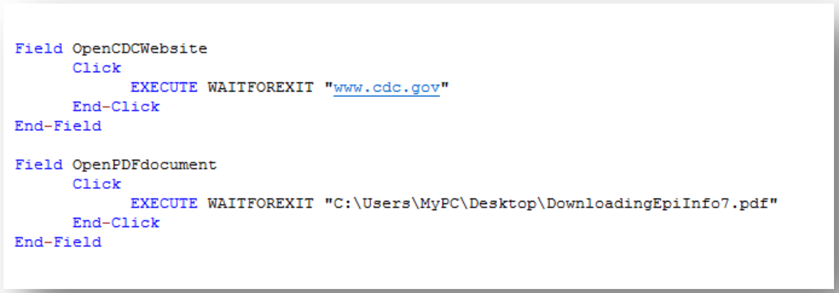
Executes a Windows program - either one explicitly named in the command or one designated within the Windows registry as appropriate for a document with the file extension that is named. This provides a mechanism for bringing up whatever program is the default on a computer without first knowing its name. The EXECUTE command accepts a series of paths, separated by semicolons, as in:

EXECUTE c:\Users\MyPC\myfile.xls;d:\myfile.xls

If the first is not found, the others are tried in succession. In Check Code, the EXECUTE command can be placed in any command block, but is often used with a command button. Check code syntax is executed when the user clicks on the command button.
EXECUTE WAITFOREXIT "<filename>"EXECUTE NOWAITFOREXIT "<filename>"

- The <filename> represents the path and program name for .exe (filename for registered Windows programs) and .com (filename -DOS binary executables) files.
- The <command-line parameters> represent any additional command-line arguments that the program can accept.
- When **Wait for Exit** command is specified (modal), the command should run and Enter should continue running. When **No Wait for Exit** command is specified (non-modal), Enter should wait until the executed program closes before continuing. When EXECUTE is run modally, permanent variables are written before the command is executed and reloaded after the command is executed.

If the example below, the CDC website page is opened when the user clicks on a command button called *OpenCDCWebsite*. A .pdf file is automatically opened when the user clicks on a command button called *OpenPDFdocument*.



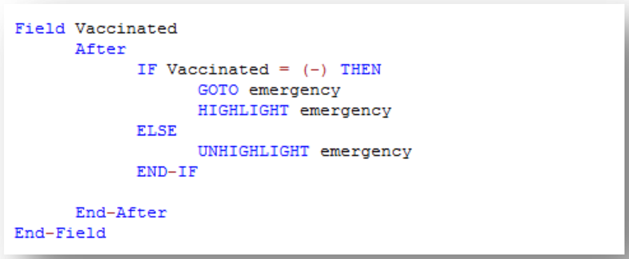
```
Field OpenCDCWebsite
      Click
            EXECUTE WAITFOREXIT "www.cdc.gov"
      End-Click
End-Field

Field OpenPDFdocument
      Click
            EXECUTE WAITFOREXIT "C:\Users\MyPC\Desktop\DownloadingEpiInfo7.pdf"
      End-Click
End-Field
```

Figure 3.58: Check code syntax for EXECUTE command

Help

The Help command allows you to pop up a help window containing a message, or even a window on a large file that allows the user to move from one block of text to another by choosing highlighted portions of the text. This is a simple form of what is known as "hypertext."

Highlight/Unhighlight This command emphasizes the location of a field by highlighting in bright yellow, for example, if a data entry error was detected. It also unhighlights the field if needed. In the example below, the field Emergency is highlighted if the response to the field Vaccinated = "No". All vaccinated questions are skipped and the field "Did you visit the emergency room?" is highlighted.

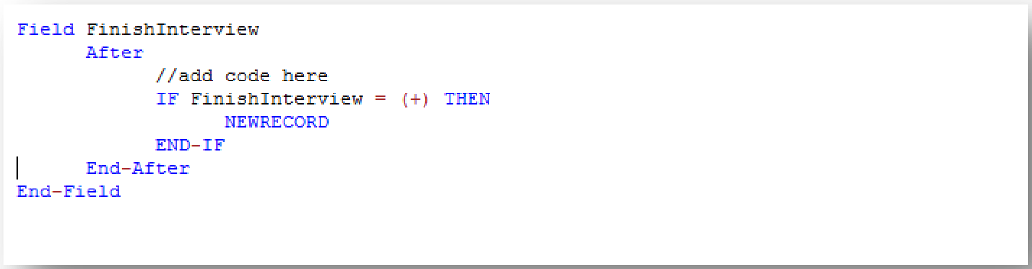


```
Field Vaccinated
      After
            IF Vaccinated = (-) THEN
                  GOTO emergency
                  HIGHLIGHT emergency
            ELSE
                  UNHIGHLIGHT emergency
            END-IF

      End-After
End-Field
```

Figure 3.59: Check code syntax for HIGHLIGHT/UNHIGHLIGHT command

New Record

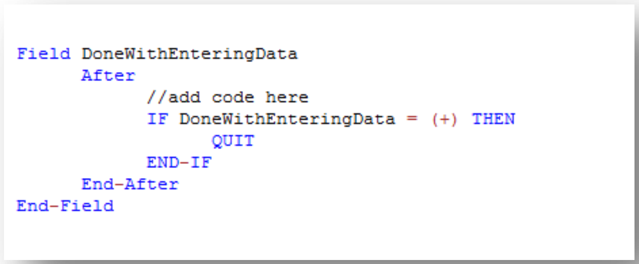
This command saves the current records data and opens a new record for data entry.



```
Field FinishInterview
      After
            //add code here
            IF FinishInterview = (+) THEN
                  NEWRECORD
            END-IF
      End-After
End-Field
```

Figure 3.60: Check code syntax for NEWRECORD command

Quit This command allows the saving of the current record and closing of the Enter application.

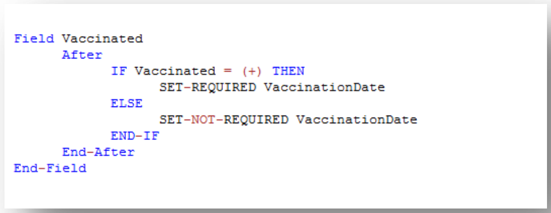


```
Field DoneWithEnteringData
      After
            //add code here
            IF DoneWithEnteringData = (+) THEN
                  QUIT
            END-IF
      End-After
End-Field
```

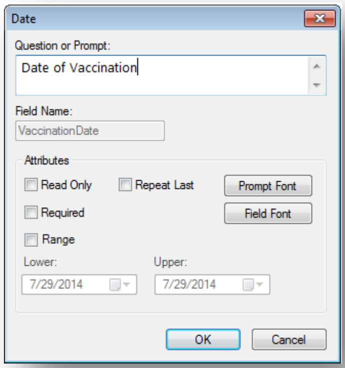
Figure 3.61: Check code syntax for QUIT command

Set-Required-Set Not Required-

Sometimes users might want to set up a field as required only if a specific criterion is met. As with the REQUIRED property, if a field is set to required during data entry through check code, the Enter module will not allow further page navigation until a value has been entered into the field. In the example below, the field VaccinationDate is set as required if the Vaccination question is answered as *Yes* using the **Set-Required** command. Notice that the REQUIRE property has not been set during the creation of the field but it has been set with the check code. Once the criterion is met, the field is set back to its original state by using the **Set-Not-Required** command.



```
Field Vaccinated
      After
            IF Vaccinated = (+) THEN
                  SET-REQUIRED VaccinationDate
            ELSE
                  SET-NOT-REQUIRED VaccinationDate
            END-IF
      End-After
End-Field
```



Date

Question or Prompt:

Date of Vaccination

Field Name:

VaccinationDate

Attributes

- [ ] Read Only
- [ ] Repeat Last
- [ ] Required
- [ ] Range

Lower: 7/29/2014

Upper: 7/29/2014

Figure 3.62: Check code syntax for SET REQUIRED command

Check Features Covered Elsewhere- The **Enter** tool can be programmed to do many interesting and complex operations not discussed in this chapter, including mathematical and logical operations with more than one field, popping up help windows, and calling programs in other languages that act on the contents of fields during data entry.

How to use EpiWeek Function

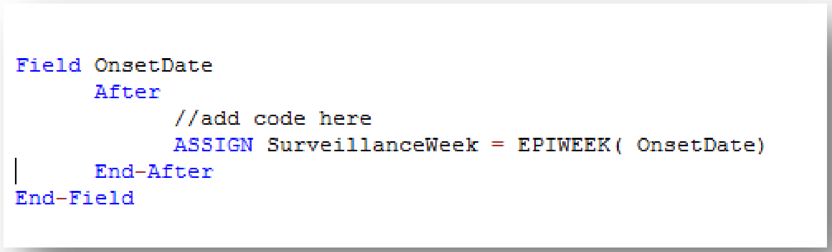
Epidemiological weeks are usually complete weeks. The ministries of health around the world define the day of the week as the first epidemiologic day. As a result, some countries may consider Sunday as the first day of the week while others may consider the first day of the week to be either Saturday or Monday.

By default, Epi Info™ 7 marks Sunday as the beginning of the epidemiological week. However, the parameter for the Epiweek function can be modified in order to change the beginning of the epidemiological week as desired.

If the year of occurrence is not relevant, use the EpiWeek method instead. The advantage of using EpiWeek is that the value is returned as a number and it can be stored in a numeric field. EpiWeek takes one required parameter that must be a date. The week is calculated relative to the year of the date provided. The Epidemiological week is calculated using the following code:

```
ASSIGN MyEpiWeek = EPIWEEK( <start_date>, {<first_day_of_week>} )
```

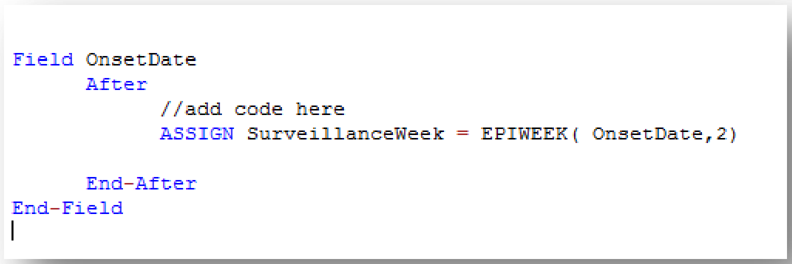
The example below shows the check code needed for automatically calculating the corresponding Epi Week into a field called *SurveillanceWeek* based on the value entered in a date field called *OnsetDate*.



```
Field OnsetDate
      After
            //add code here
            ASSIGN SurveillanceWeek = EPIWEEK( OnsetDate)
      End-After
End-Field
```

Figure 3.63: Check code syntax for EPIWEEK function

In the other hand, if you are required to modify the first day of the week parameter because the week does not start on a Sunday, the check code syntax would need to be updated. In the example below, the epidemiological week will be calculated based on a starting day of Monday (i.e. Sunday will be 1, Monday will be 2, Tuesday will be 3, etc.)



```
Field OnsetDate
	After
		//add code here
		ASSIGN SurveillanceWeek = EPIWEEK( OnsetDate,2)

	End-After
End-Field
```

Figure 3.63: Check code syntax for EPIWEEK function with beginning week parameter

Proper Check Code Syntax

Using the proper check code syntax is important based on the field type. Here are some examples of the proper syntax to use based on the field type;

- Assign the 'Age' field (numeric field type) the value 24
  - **ASSIGN Age = 24**
- Assign the 'Ill' field (Yes/No field type) the value No
  - **ASSIGN Ill = (-)**
- Assign the 'AteChicken' field (checkbox field type) the value Yes
  - **ASSIGN AteChicken = (+)**
- Assign the 'DateOfInterview' field (Date field type) the value 5/5/2012
  - **ASSIGN DateOfInterview = 5/5/2012**
- Assign the 'CaseStatus' field (legal values field type) the value "Confirmed"
  - **ASSIGN CaseStatus = "Confirmed"**
- Assign the 'Gender' field (Comment Legal field) the value "F". The field is coded as M-Male, F-Female.
  - **ASSIGN Gender = "M"**